

Frontiers
in
Artificial
Intelligence
and
Applications

NEW TRENDS IN SOFTWARE METHODOLOGIES, TOOLS AND TECHNIQUES

Edited by
Hamido Fujita
Paul Johannesson

IOS
Press

Ohmsha

VISIT...

LANZAROTE
Caliente.COM

NEW TRENDS IN SOFTWARE METHODOLOGIES, TOOLS AND TECHNIQUES

Frontiers in Artificial Intelligence and Applications

*Series Editors: J. Breuker, R. López de Mántaras, M. Mohammadian, S. Ohsuga and
W. Swartout*

Volume 84

Previously published in this series:

- Vol. 83, V. Loia (Ed.), *Soft Computing Agents*
- Vol. 82, E. Damiani et al. (Eds.), *Knowledge-Based Intelligent Information Engineering Systems and Allied Technologies*
- Vol. 81, In production
- Vol. 80, T. Welzer et al. (Eds.), *Knowledge-based Software Engineering*
- Vol. 79, H. Motoda (Ed.), *Active Mining*
- Vol. 78, T. Vidal and P. Liberatore (Eds.), *STAIRS 2002*
- Vol. 77, F. van Harmelen (Ed.), *ECAI 2002*
- Vol. 76, P. Šinčák et al. (Eds.), *Intelligent Technologies – Theory and Applications*
- Vol. 75, I.F. Cruz et al. (Eds.), *The Emerging Semantic Web*
- Vol. 74, M. Blay-Fornarino et al. (Eds.), *Cooperative Systems Design*
- Vol. 73, H. Kangassalo et al. (Eds.), *Information Modelling and Knowledge Bases XIII*
- Vol. 72, A. Namatame et al. (Eds.), *Agent-Based Approaches in Economic and Social Complex Systems*
- Vol. 71, J.M. Abe and J.I. da Silva Filho (Eds.), *Logic, Artificial Intelligence and Robotics*
- Vol. 70, B. Verheij et al. (Eds.), *Legal Knowledge and Information Systems*
- Vol. 69, N. Baba et al. (Eds.), *Knowledge-Based Intelligent Information Engineering Systems & Allied Technologies*
- Vol. 68, J.D. Moore et al. (Eds.), *Artificial Intelligence in Education*
- Vol. 67, H. Jaakkola et al. (Eds.), *Information Modelling and Knowledge Bases XII*
- Vol. 66, H.H. Lund et al. (Eds.), *Seventh Scandinavian Conference on Artificial Intelligence*
- Vol. 65, In production
- Vol. 64, J. Breuker et al. (Eds.), *Legal Knowledge and Information Systems*
- Vol. 63, I. Gent et al. (Eds.), *SAT2000*
- Vol. 62, T. Hruška and M. Hashimoto (Eds.), *Knowledge-Based Software Engineering*
- Vol. 61, E. Kawaguchi et al. (Eds.), *Information Modelling and Knowledge Bases XI*
- Vol. 60, P. Hoffman and D. Lemke (Eds.), *Teaching and Learning in a Network World*
- Vol. 59, M. Mohammadian (Ed.), *Advances in Intelligent Systems: Theory and Applications*
- Vol. 58, R. Dieng et al. (Eds.), *Designing Cooperative Systems*
- Vol. 57, M. Mohammadian (Ed.), *New Frontiers in Computational Intelligence and its Applications*
- Vol. 56, M.I. Torres and A. Sanfeliu (Eds.), *Pattern Recognition and Applications*
- Vol. 55, G. Cumming et al. (Eds.), *Advanced Research in Computers and Communications in Education*
- Vol. 54, W. Horn (Ed.), *ECAI 2000*
- Vol. 53, E. Motta, *Reusable Components for Knowledge Modelling*
- Vol. 52, In production
- Vol. 51, H. Jaakkola et al. (Eds.), *Information Modelling and Knowledge Bases X*
- Vol. 50, S.P. Lajoie and M. Vivet (Eds.), *Artificial Intelligence in Education*
- Vol. 49, P. McNamara and H. Prakken (Eds.), *Norms, Logics and Information Systems*
- Vol. 48, P. Návrat and H. Ueno (Eds.), *Knowledge-Based Software Engineering*
- Vol. 47, M.T. Escribá and F. Toledo, *Qualitative Spatial Reasoning: Theory and Practice*

ISSN: 0922-6389

New Trends in Software Methodologies, Tools and Techniques

Proceedings of Lyee_W02

Edited by

Hamido Fujita

Iwate Prefectural University, Iwate, Japan

and

Paul Johannesson

Royal Institute of Technology, Stockholm, Sweden



Amsterdam • Berlin • Oxford • Tokyo • Washington, DC

© 2002. The authors mentioned in the Table of Contents

All rights reserved. No part of this book may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, without prior written permission from the publisher.

ISBN 1 58603 288 7 (IOS Press)

ISBN 4 274 90543 8 C3055 (Ohmsha)

Library of Congress Control Number: 2002111981

Publisher

IOS Press

Nieuwe Hemweg 6B

1013 BG Amsterdam

The Netherlands

fax: +31 20 620 3419

e-mail: order@iospress.nl

Distributor in the UK and Ireland

IOS Press/Lavis Marketing

73 Lime Walk

Headington

Oxford OX3 7AD

England

fax: +44 1865 75 0079

Distributor in the USA and Canada

IOS Press, Inc.

5795-G Burke Centre Parkway

Burke, VA 22015

USA

fax: +1 703 323 3668

e-mail: iosbooks@iospress.com

Distributor in Germany, Austria and Switzerland

IOS Press/LSL.de

Gerichtsweg 28

D-04103 Leipzig

Germany

fax: +49 341 995 4255

Distributor in Japan

Ohmsha, Ltd.

3-1 Kanda Nishiki-cho

Chiyoda-ku, Tokyo 101-8460

Japan

fax: +81 3 3233 2426

LEGAL NOTICE

The publisher is not responsible for the use which might be made of the following information.

PRINTED IN THE NETHERLANDS

Preface

Software is the essential enabler for the new economy and science. It creates new markets and new directions for a more reliable, flexible and robust society. It empowers the exploration of our world in ever more depth.

However, software often falls short of our expectations. Current software methodologies, tools and techniques remain expensive and not yet reliable for a highly changeable and evolutionary market. Many approaches have been proven only as case-by-case oriented methods.

This book presents a number of new theories and trends in the direction in which we believe software engineering should develop to transform the role of software and science in tomorrow's information society .

This book is an attempt to capture the essence of a new state of art in software engineering and its supporting technology. The book also aims at identifying the challenges such a technology has to master. It contains papers accepted at the First International Workshop on New Trends in software Methodology and Technology, (SoMeT_02) and also named as the First Lyee International Workshop (Lyee_W02), held in Sorbonne, Paris, from 3rd to 5th October 2002, (http://www.lyee-project.soft.iwate-pu.ac.jp/lyee_w02). This workshop brought together researchers and practitioners to share their original research results and practical development experiences in software engineering, and its related new challenging technology. Lyee is a newly emerged Japanese software methodology that has been patented in many countries in Europe, Asia, and America. But it is still in its early stage of emerging as a new software paradigm. This conference and the series it initiates will elaborate on such new trends and related academic research studies and development. A major goal of this international conference was to gather scholars from the international research community to discuss and share research experiences on new software methodologies related to Lyee and non-Lyee approaches. The conference also investigated other comparable theories and practices in software engineering, including emerging technologies, from their computational foundations in terms of models, methodologies, and tools. These are essential for developing a variety of information systems research projects and to assess the practical impact on real-world software problems.

The **Lyee International Workshop (Lyee-W02)** was held on **October 3 – 5, 2002**, in Sorbonne, Paris, France. This event initiates a coming series that will include the 2nd workshop (**Lyee-W03**), to be organized in Stockholm, Sweden in October 2003 (http://www.lyee-project.soft.iwate-pu.ac.jp/lyee_w03), and the 3rd workshop (**Lyee-W04**), the 4th workshop (**Lyee-W05**) to be held in Berlin, Germany and others to follow.

This book is also a means for presenting parts of the results of the Lyee International research project (<http://www.lyee-project.soft.iwate-pu.ac.jp>), which aims at the exploration and development of novel software engineering methods and software generation tools based on the Lyee framework. This project is sponsored by Catena and other major Japanese enterprises in the area of software methodologies and technologies.

This book provides an opportunity for exchanging ideas and experiences in the field of software technology opening up new avenues for software development, methodologies, tools and techniques.

The Lyee methodology captures the essence of the innovations, controversies, challenges, and possible solutions of the software industry. This world-wide patented theory was born from long experience, and it is time through this workshop to try to let it stimulate the academic research on software engineering that will close the gap between theory and practice. We believe this book creates an opportunity for us in the software engineering community to think about where we are today and where we are going,

The book is a collection of 31 carefully selected papers.

The areas which this book covers are

- * Requirement engineering and requirement elicitation,
- * Software methodologies and Lyee oriented software techniques,
- * Automatic software generation, reuse, and legacy systems,
- * Software quality and process assessment,
- * Intelligent software systems and evolution,
- * Software optimization and formal methods,
- * Static and dynamic analysis on Lyee-oriented software performance model
- * End-user programming environment using Lyee,
- * Ontology and philosophical aspects on software engineering,
- * Business software models and other kinds of software application models, based on Lyee theory,
- * Software Engineering models.

All papers published in this book are reviewed and selected by an international program committee. Each paper has been reviewed by three or four reviewers and has been revised based on the review reports. The acceptance rate is 56%. The papers were reviewed on the basis of technical soundness, relevance, originality, significance, and clarity.

This book was made possible by the collective efforts from all the Lyee International project collaborators and other people and supporters. We gratefully thank Iwate Prefectural University, the sponsors, and others for their overwhelming support.

This book will be the 1st milestone in mastering new challenges on software and its new promising technology, within SoMeT_02 or Lyee_W02 framework and others. Also, it gives the reader new insights, inspiration and concrete material to elaborate and study this new technology.

The Editors

Program Chairs

Hamid (Issam) Fujita, Iwate Prefectural University, Iwate, Japan

e-mail: issam@soft.iwate-pu.ac.jp

Paul Johannesson, Royal Institute of Technology, Stockholm, Sweden

e-mail: pajo@dsv.su.se

Program Committee

Norio Shiratori, Tohoku University, Japan

Teruo Higashino, Osaka University, Osaka

Kasem Saleh, American University of Sharjah, UAE

Ridha Khedri, McMaster University, Hamilton, Canada

Marite Kirikova, Riga Technical University, Latvia

Remigijus Gustas, Karlstad University, Sweden

Love Ekenberg, Mid Sweden University Sweden,

Benkt Wangler, University of Skovde, Sweden

Rudolf Keller, Zuehlke Engineering AG, Switzerland

Mohamed Mejiri, Laval University, Quebec, Canada

Anna-Maria Di Sciullo, University de Quebec de Montreal, Canada

Soundar Kumara, The Pennsylvania State University, USA

Margeret M Burnett, Oregon State University, USA

Michael Oudshoorn, University of Adelaide, Australia

Isaac Marks, King's College London, London, UK

Ernest Edmonds, Loughbrough University, UK

Stuart Toole, University of Central England, UK

Nicole Levy, Universite de Versailles St-Quentin en Yvelines, France

Christophe Sibertin-Blanc, Toulouse_1 University, France

Victor Malyshkin, Russian Academy of Sciences, Russia

Volker Gruhn, Dortmund University, Germany

Vincenzo Ambriola, Pisa University, Italy

Roberto Poli, Terent University, Italy

Domenico M. Pisanelli, ITBM - CNR, Rome, Italy

Liliana Albertazzi, Mitteleuropa foundation Research Centre, Bolzano, Italy

Samuel T. Chanson, Hong Kong Univ. of Science & Technology, China

Organizing Chairs

Selmin Nurcan, University de Paris_1-Pantheon Sorbonne, Paris, France

e-mail : nurcan.iae@univ-paris1.fr

Benkt Wangler, University of Skovde, Sweden

e-mail : benct.wangler@ida.his.se

Tutorial Organizing Chairs

Roberto Poli, Trento University, Italy

e-mail : roberto.poli@soc.unitn.it

Shigeru Tomura, (ICBSMT), Japan

e-mail : s-tomura@lyee.co.jp

General Chairs

Colette Rolland

University de Paris_1-Pantheon Sorbonne, Paris, France

Fumio Negoro

Institute of Computer Based Software Methodology and Technology (ICBSMT), Japan

Additional Reviewers

Masaki Kurematsu

Iwate Prefectural University, Iwate, Japan

e-mail:kure@soft.iwate-pu.ac.jp

Jun Hakura

Iwate Prefectural University, Iwate, Japan

e-mail:hakura@soft.iwate-pu.ac.jp

Bipin Indurkha

Tokyo University of Agriculture and Technology, Tokyo, Japan

e-mail: bipin@cc.tuat.ac.jp

Contents

Preface	v
Conference Organisation	vii
Invited presentation (1)	
Lyee's Hypothetical World, <i>Fumio Negoro</i>	3
Chapter 1. Intention and Hypothetical Oriented Software	
Intention as Architecture of Spaces, <i>Liliana Albertazzi</i>	25
Requirement Shifters, <i>Roberto Poli</i>	35
On the Philosophical Foundation of Lyee: Interaction Theories and Lyee, <i>Bipin Indurkha</i>	45
Chapter 2. Software Architecture and Software Intentional Models	
Architecture Underlying the Lyee Method: Analysis and Evaluation, <i>A. Ramdane-Cherif, L. Hazem and N. Levy</i>	55
A Word-unit-based Program: Its Mathematical Structure Model and Actual Application, <i>Osamu Arai and Hamido Fujita</i>	63
A Top-Down Approach to Identifying and Defining Words for Lyee Using Condition Data Flow Diagrams, <i>Shaoying Liu</i>	75
A Linguistic Method Relevant for Lyee, <i>Gregers Koch</i>	88
Using Natural Language Asymmetries in Information Processing, <i>Anna Maria Di Sciullo</i>	96
Chapter 3. Lyee-Oriented Software Engineering and Applications	
Engineering Characteristics of Lyee Program, <i>Ryosuke Hotaka</i>	109
Ontologies and Information Systems: the Marriage of the Century? <i>Domenico M. Pisanelli, Aldo Gangemi and Geri Steve</i>	125
The Key Features of the LyeeAll Technology for Programming Business-like Applications, <i>Victor Malyshkin</i>	134
Software Process of the Insurance Application System by Using Lyee Methodology, <i>Volker Gruhn, Raschid Ijoui, Dirk Peters, Clemens Schäfer, Takuya Kawakami and Makoto Asari</i>	141
Invited Presentation (2)	
A User Centric View of Lyee Requirements, <i>Colette Rolland</i>	155
Chapter 4. Requirement Engineering and Meta Models	
VOLYNE: Viewpoint Oriented Engineering of the Requirement Elicitation Process for Lyee Methodology, <i>Pierre-Jean Charrel, Laurent Perrussel and Christophe Sibertin-Blanc</i>	173
Towards an Understanding of Requirements for Interactive Creative Systems, <i>Michael Quantrill</i>	187
Generating Lyee Programs from User Requirements with a Meta-model Based Methodology, <i>Carine Souveyet and Camille Salinesi</i>	196
Lyee Program Execution Patterns, <i>Mohamed Ben Ayed</i>	212

Invited Presentation (3)

Bringing HCI Research to Bear Upon End-User Requirement Specification. <i>Margaret Burnett</i>	227
---	-----

Chapter 5. Human Factor and User Intention Capturing in Software

End-User Testing of Lyee Programs: A Preliminary Report. <i>Darren Brown, Margaret Burnett and Gregg Rothmel</i>	239
Developing Multimedia Systems: the Understanding of Intention. <i>Anthony Burgess</i>	254
Capturing Collective Intentionality in Software Development. <i>Benkt Wangler and Anne Persson</i>	262

Chapter 6. Enterprise Software Models and Software Engineering

Extending Lyee Methodology using the Enterprise Modelling Approach. <i>Remigijus Gustas and Prima Gustienė</i>	273
Extending Process Route Diagrams for Use with Software Components. <i>Lars Jakobsson</i>	289

Chapter 7. Software Process Model and Configuration Management

The Lyee Base Process in Framework. <i>Vincenzo Ambriola, Giovanni A. Cignoni and Hamido Fujita</i>	303
Configuration Management Concept for Lyee Software. <i>Volker Gruhn, Raschid Ijoui, Dirk Peters and Clemens Schäfer</i>	317
An Experiment on Software Development of Hospital Information System . <i>Yutaka Fumyu, Jun Sasaki, Takushi Nakano, Takayuki Yamane and Hiroya Suzuki</i>	328

Chapter 8. Automatic Software Generation and Requirement Verification

Describing Requirements in Lyee and in Conventional Methods: Towards a Comparison. <i>Gregor v. Bochmann</i>	343
Automated Word Generation for the Lyee Methodology. <i>Benedict Amon, Love Ekenberg, Paul Johannesson, Marcelo Munguanaze, Upendo Njabili and Rika Manka Tesha</i>	357
Static Analysis on Lyee Oriented Software. <i>Mohamed Mejri, Béchir Ktari and Mourad Erhioui</i>	375
EFLE: An Environment for Generating Lingware Systems Code from Formal Requirements Specification. <i>Bassem Bouaziz, Bilel Gargouri, Mohamed Jmaiel and Abdelmajid Ben Hamadou</i>	395
Author Index	405

Invited Presentation 1

This page intentionally left blank

Lyee¹'s Hypothetical World

Fumio NEGORO

*The Institute of Computer Based Software Methodology and Technology,
11-3 Takanawa 3-chome, Minato-ku, Tokyo 108-0074, Japan*

Abstract. We think that determining software requirements follows the same process as that of formulating something existent. Unless we consider the process of existence and utilize it as a norm, it would be next to impossible to overcome the problems brought up by determination of development requirement.

Suppose existence is composed of cause and result. If we trace the existence back to the cause from the result in a reversible manner, we will be led to unrecognizable world in the end. We know we experience this on a daily basis so that we naturally do not think of it anymore. This suggests, however, how clearly we can recognize things is determined by the relationship between the recognizable and the unrecognizable worlds.

This is a basic notion of our study. This paper aims to find a way to axiomatically define 1) recognizable existence, 2) unrecognizable existence, and 3) relationship between these two.

1. The Observer and the Observed

In our study, existence is defined as a spatial expansion (hereinafter simply called space) that is accompanied by description. Space governs a location of existence, while its description represents characteristics of its space. These characteristics are, for example, assumed as follows: 1) whole, 2) part, 3) non-existence, 4) existence, 5) the observed (others), and 6) observer (self). They are defined with what we call links. A link is defined with a pair of a substance and attributes. The attributes are represented by a set, while the substance is one element to be determined with the attributes. The substance represents something existent, while the attributes take a role of describing a nature of the substance. One element representing the substance and other elements representing the attributes are called logical atoms. Details of this atom will be explained later.

Links are categorized into several types, since attributes of a link form a set and the set transforms itself. In other words, the concept of set is changing. In the process of this change, what type of link is in question is determined. The transition starts from the most primitive phase in terms of definition of a set. Following more complicated definition phases step by step, links are to be determined. This process is governed by deterministic rules.

When a concept of set that determines a link cannot be deepened furthermore, such a state of link is called singular link. This singular link is formed to describe the five mechanisms: 1) intention, 2) consciousness, 3) objectification, 4) memory, and 5) assimilation. An observer refers to this singular link that is explained by these five mechanisms. Provided, however, that description of the five mechanisms is not made for other links that are formed in the process of reaching a singular link.

The mechanism of objectification is built, for example, to create existence that has a mass from something that does not possess a mass. The existence that has a mass is called something

¹ Lyee, standing for the governmental methodology for software providence, refers to the comprehensive method to define what is software in the software field.

existent, while the existence that does not have a mass is called something non-existent. Singular links as well as other links leading to them are all considered non-existence.

Something existent constructs the mechanism of memory, thereby producing another existence. That existence itself is referred to as the observed.

In the same manner, something existence constructed the mechanism of assimilation, thereby producing another existence. But that existence can never be the observed. What we emphasize here is that an observer can never be the observed.

It is the observed except him/herself that the observer can observe or recognize.

Let me paraphrase it. It is impossible for my being to observe what I am. It is possible, however, for my being to observe my feelings or an apple on the table. That is to say, "I" am the observer in this context. "I" am a singular link in the theory. In this example, my feelings and an apple are considered the observed. They are natural links in the theory.

The five mechanisms are shown in Figure 2. When they are expressed in a model, that is called three-dimension-like space model abbreviated as TDM. TDM can be said as a model of the observer (see Figure 4).

Let us apply the idea of the observer and the observed to software. Suppose the observed is software development requirement, TDM will naturally be its observer, that is a program.

The reason why the Lyee theory can be a development methodology of software lies in that 1) TDM is able to express a program and 2) thinking process to define TDM is deterministically determined by the definition given by this study.

Judging from the process of formation of TDM, it is obvious that a program structure of TDM is not a logic-based program made in a conventional manner. TDM's program has a fundamentally different structure from that of a program, which is produced by conventional development methodology. Our web-site shows the difference between Lyee methodology and the other conventional methodologies.

2. Axioms

In order to lay a ground to define transitional process of a set that is composed of attributes, our study formulates three axioms:

Axiom 1: A space (03) described by the "part (01)" and the "whole (02)" is something existent or existence (04).

Axiom 2: The existence is coupling (05).

Axiom 3: The existence is objectified (06).

These axioms are provided to give a definition to existence with six terms. Based on these, it becomes possible to define the process from the observed to the observer without causing any contradiction.

This process also gives definition to the five mechanisms. If the performance of these mechanisms is verified, those definitions can be set as theorems that are constructed on the basis of the axioms. The verification is conducted in software field.

2.1. Terms to be used for axioms

Terms to be used for the axioms are necessarily defined to discuss the unrecognizable world. Meaning of the axioms is therefore different from ordinarily accepted axioms.

01. Part

Substance and attributes formulate a pairing relation, while the substance represents existence and the attributes describe the substance. This relation is called link in our study. In the case of natural link, one cognitive atom is chosen for the substance from outside, the one of which spatial notion is closest to the largest possible occupying space of the attribute. As for cognitive atoms, they will be later explained in details.

The fact that the substance representing the existence itself is chosen from the outside of the attributes means the attributes are forming a part toward the substance. From this concept, we can say that a natural link is to discuss the part of existence.

A spatial notion is defined as an expansion of space inherent in a cognitive atom. An occupying space of the attribute represents a total of spatial notions of cognitive atoms, since the attribute forms a set of elements that are cognitive atoms.

02. Whole

In the case of consciousness link, one consciousness atom is chosen for the substance from the consciousness atoms that belong to the attribute. The chosen consciousness atom has a spatial notion that should be closest to the smallest possible occupying space of the attribute. Consciousness atoms will be clearly explained later.

The fact that the substance representing the existence itself is chosen from the attributes that has a pairing relation with the substance means the attribute is forming the whole toward the substance. From this concept, we can say that a consciousness link is to discuss the whole of existence.

03. Space

It refers to a spatial expansion of the substance, indicating a spatial notion.

04. Existence

Existence is represented by a spatial notion of the substance and described by its attributes.

05. Coupling

Natural links are being coupled with each other to create further existence. The existence created by the coupling operation is also represented by one natural link.

06. Objectification

One of the objectification operations is, for example, to change a spatial notion having no mass into the one having a mass. This operation can be applied to a natural link.

2.2. Significance of Axioms

Axiom 1 states that existence is expressed with space and it is described with part and whole. Description of the part is made with a natural link, whereas the description of the whole is formed in a consciousness link.

An apple that I can observe is in fact an apple that is considered as a part to be described by a natural link. Due to the definition of the five mechanisms, however, it is impossible for me to observe an apple as a whole to be described by a consciousness link. Even though the apple is only one apple that can be held in my hand, I am not allowed to observe it as the whole of existence.

Observation is formulated toward the description of the part. If the whole can be observed, that means description of the part has no limits to approach the description of the whole as much as possible. Such an observation to be formed, however, is impossible to define the five mechanisms. Thus, existence can be observed only as the part. In this case, the description of the whole is included by the description of the part so that it is out of the target of observation. Observation could be understood as cognition.

Axiom 2 states that existence forming a part is to be represented by a set. In other words, natural links becomes a set. In this study, formation of a set is called coupling. For example,

one piece of machinery can be a part, at the same time be a whole. When they are composed to something existent, say, an automobile, it can also be a part as well as a whole.

Axiom 3 states that there are two types of existences: one does not have a mass, and the other has a mass. In our study, to have a mass is called objectification. The former also refers to something non-existent, and the latter something existent. Existence that can be a target of objectification is only a natural link. Let us give you an example to explain this. An apple on the table and an apple that appears in our minds when we look at it are both objectified natural links, that is, something existent. But there is another natural link, which is present before its objectification. This refers in fact something non-existent, which plays a role of building the mechanism of consciousness. In this case, the substances of these two natural links are the same, but their attributes are different.

Natural links prior to objectification entail links for expressing a process of their formation, namely, probabilistic link, consciousness link, normalization link, and multiplex link. Objectified natural link entails another natural link that is existent before objectification.

3. Five Mechanisms

Five mechanisms which are described with a singular link are used to define an observer. This definition is summarized in Figure 1.

- 1) With the work of coupling of natural link (01), a singular link (04) is established, which has a boundary atom (02) as its substance (03). On the other hand, a critical link (05) is normally established as previously mentioned, which has a boundary atom as its substance.
- 2) Since a singular link and a critical link share the boundary atom as the substance of its own, a certain corresponding relation (06) is formed between the two.
- 3) That means that a corresponding relation between the whole and the part which constitute one existence is constructed. This relation refers to "the mechanism of intention" which is built in the singular link.

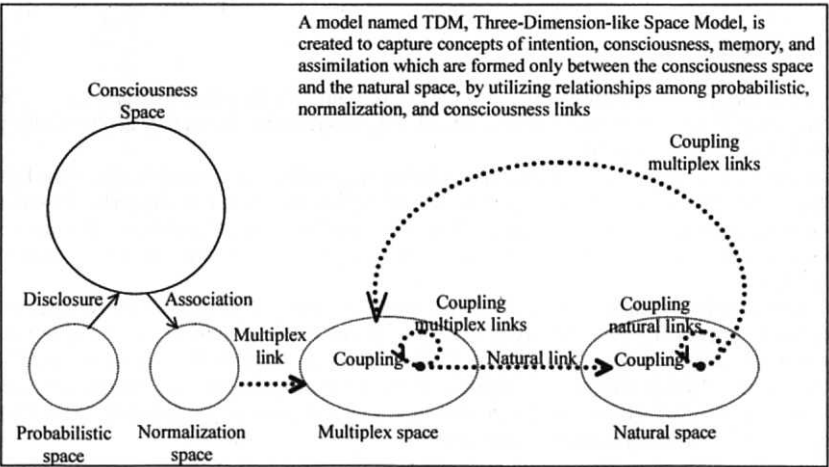


Figure 1. Relations among Spaces created by the Ideal Space

- 4) Once the mechanism of intention is constructed, a critical link chooses one natural link from the history of coupling (07) that reaches the singular link. The selected natural link has attributes of cognitive atoms (10) of which number is less than the number of the consciousness atoms (09) by one, which belong to the attributes (08) of the critical link.
- 5) If one and more multiplex links (11) are formed of which substance is a cognitive atom belonging to the attributes of the chosen natural link, this relation refers to "the mechanism of consciousness" which is constructed in the singular link.
- 6) If there are same cognitive atoms as those that belong to the attributes of the chosen natural link in the cognitive atoms belonging to the attributes of the multiplex link, the singular link produces another link which has a natural atom of the newly chosen natural link as its substance and attributes that are a collection of the same cognitive atoms. This refers to "the mechanism of objectification" to be objectified in this singular link. It is this natural link that is an observer.
- 7) Spatial notions of the cognitive atoms that are attributes of a natural link produced by the mechanism of objectification turns from non-existence (12) to existence (13). In the supposition of our study, what is non-existence refers to a spatial notion which does not have a mass, whereas what is existence refers to a spatial notion that has a mass.
- 8) Reversible (14) operation does not work in the mechanism of objectification. Its reverse objectification is not formed. The whole state of the existence is to be described with consciousness link, whereas the partial state of the non-existence is to be described with probabilistic (15), normalization (16), multiplex (17), and natural links. The existence is to be described with objectified natural link.
- 9) If the number of cognitive atoms belonging to the attributes of the objectified natural link is less than the number of cognitive atoms belonging to the attributes of the chosen natural link, this relation refers to "the mechanism of memory" that is established in the singular link.
- 10) If the number of cognitive atoms belonging to the attributes of the objectified natural link is the same and more, this relation refers to "the mechanism of assimilation" that is established in the singular link.

3.1. Terms for Defining an Observer

Although some of the terms mentioned below are already used in this paper, the following terms are mandatory to define an observer. The purpose of the hypothesis of our study can be said as definition of these terms.

01. Natural link
02. Boundary atom
03. Substance
04. Singular link
05. Critical link
06. Corresponding relation
07. History of coupling
08. Attributes
09. Consciousness atom
10. Cognitive atom
11. Multiplex link
12. Non-existence, something non-existent
13. Existence, something existent
14. Reversible
15. Probabilistic link

- 16. Normalization link
- 17. Multiplex link

4. Hypothesis

In our study, a transitional process to reach a singular link as well as the operation of the five mechanisms are considered the work of "providence." This providence is defined on a premise of three hypothetical axioms. More specifically speaking, to define the providence is in actuality to define the above-mentioned terms.

4.1. Time Velocity

001: Fundamental and discrete elements to form consciousness and cognitive elements are hypothetically set up and named time velocity. Time velocities are rational numbers that cannot be recognized.

4.1.1. Time Velocity, Unknown Space (UKS), Ideal Space (IDS), and Logical Atoms

A basic structure to generate links based on time velocity is constructed in a hypothetical manner. It is called ideal space and abbreviated as IDS (see Figure 2).

002: Unknown space abbreviated as UKS refers to a whole set of time velocities.

003: Subsets of UKS are not formed.

004: All time velocities in UKS are assumed to exist at the very beginning.

005: Total of time velocities in UKS is denoted by $\Sigma_1 \Phi V_i$. This value is invariable and represents energy of UKS. But a time velocity of this value does not exist in UKS, which is called a singular number. Φ herein is the maximum natural number that we cannot recognize. The inverse of $\Sigma_1 \Phi V_i$ is defined as 2ϵ . Its value is the minimum time velocity (V_m) among those existent in UKS. This also indicates a size of a spatial expansion of UKS.

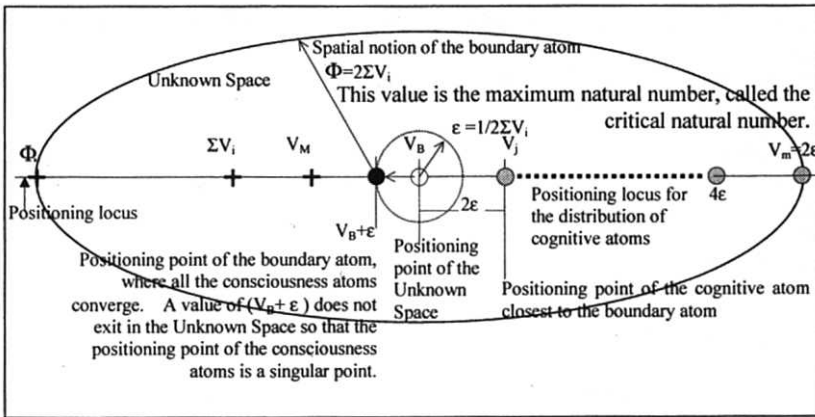


Figure 2. Structure of the Ideal Space

- 006: The inverse of value ϵ is defined as $2\Sigma_1^{\Phi}V_i$, which is universal and is denoted by Φ .
- 007: The value ϵ represents a diameter of the spatial notion of UKS. As this value does not exist in UKS, it is a singular number.
- 008: Any time velocity does not exist between the maximum natural number (Φ) and the maximum time velocity (V_M).
- 009: A difference between any time velocity and its next-placed one in UKS is 2ϵ .
- 010: In UKS, time velocities are consecutively generated one after another with an irregular interval. The value of a newly generated time velocity is already present in UKS so that it is overlapped with that of an old one.
- 011: Overlapped time velocities are not allowed to co-exist in UKS so that a newly generated time velocity has to be projected onto another space, that is IDS.
- 012: If time velocities are overlapped in IDS, however, the newly generated time velocity is to be projected onto another IDS to avoid such overlapping.
- 013: When the number of time velocities becomes more than three, one of them is chosen to be a representative time velocity, that is called boundary atom.
When all time velocities are positioned in a descending order, an atom placed in the middle of the positioning locus is designated as a boundary atom. When the number of time velocities is even, a time velocity which stands in a position of an odd number plus one is assigned to be the boundary atom. The reason why all time velocities are placed always in a descending order is that they are governed by the energy of UKS.
- 014: Every time a new time velocity is projected onto IDS, the boundary atom is redefined. This is a natural process, judging from the definition of a boundary atom. Once the boundary atom is determined, the position energies and spatial notions of time velocities are to be redefined as well. These values are also rational numbers belonging to UKS, with detailed explanation to be later given.
- 015: Logical atom is composed of time velocity, position energy, and spatial notion.
- 016: Suppose time velocity is denoted by V_i , its position energy and spatial notion are defined as follows:
1) Position energy of $V_i = V_B - V_i$
 V_B takes a role to make relative a time velocity of the boundary atom, while a position energy makes relative the time velocity.
2) A spatial notion of time velocity $V_i = 1/(V_B - V_i)$.
- 017: If $V_B - V_i > 0$, its logical atom becomes a consciousness atom.
- 018: If $V_B - V_i < 0$, its logical atom becomes a cognitive atom.
- 019: A consciousness atom is positioned with its own time velocity being a coordinate, as long as there is a time velocity of which value is the same as that of its spatial notion in IDS. Unless otherwise, it is afloat in IDS.
- 020: Once the boundary atom is determined, UKS is unconditionally positioned having a time velocity of the boundary atom that becomes its coordinate, because UKS has an invariable spatial notion (2ϵ).
- 021: With this, the boundary atom is unconditionally positioned having its own time velocity + ϵ which becomes its coordinate. The value is a singular number. The boundary atom is given a right to position there.
- 022: Position energy and spatial notion of the boundary atom are defined as follows:
1) Position energy = ϵ
2) Spatial notion = $1/\epsilon$
The position energy of the boundary atom is not zero, but a value ϵ . $1/\epsilon$ is a singular number and equals to the maximum natural number (Φ). Because of this, IDS is to be replaced by a spatial notion of the boundary atom.

- 023: The boundary atom is regarded as a consciousness atom.
- 024: The boundary atom has a bigger time velocity than its original one by a value ϵ . This increased value is provided by convergence of time velocities of all consciousness atoms. As a result, all consciousness atoms are unconditionally positioned with the same coordinates as that of the boundary atom.
Therefore, time velocities of all consciousness atoms are turned into V_B . But their position energies and spatial notions are decided before their positioning. That is, the consciousness atom of time velocity V_i is defined as follows: $[V_B, V_B - V_i < 0, 1/(V_B - V_i)]$.
- 025: A set of coordinates of cognitive atoms, consciousness atoms, and UKS forms a positioning locus or line.
Every time a new time velocity comes to IDS, that affects to redefine the boundary atom, position energies, spatial notions, and positioning conditions. For the adjustment of redefinition, all consciousness and already positioned cognitive atoms will leave the positioning locus. That means the positioning line disappears. When the positioning locus disappears, UKS will not be allowed to stay on the positioning line so that it will also leave its IDS.
On the other hand, since the spatial notion representing IDS is invariable, it will never disappear once it is created. Thus, all consciousness and cognitive atoms leaving the positioning line are allowed to stay afloat in the spatial notion of IDS.
When a boundary atom and logical atoms are redefined, however, the positioning locus is re-established and UKS also appears again to be placed on the positioning locus. Nevertheless, there are cognitive atoms which are not allowed to be placed on the positioning line. Those will be afloat in the spatial notion of IDS until the next opportunity for positioning comes.
- 026: Time velocities coming to IDS will never disappear.

4.2. Logical Atoms

Logical atoms are elements that constitute a link.

4.2.1. Consciousness λ Set, Consciousness λ Column, Consciousness Link, Critical Link, and Consciousness Space

- 027: The greatest possible number of sets, of which elements are consciousness atoms, is made in IDS. Such a set is named consciousness λ set. The consciousness atoms of these sets are to be shared by the consciousness λ set. These atoms do not exist for each set. An outline of the consciousness λ set produces some patterns on the positioning locus.
- 028: If the number of consciousness atoms is p , take λ pieces of different consciousness atoms from the p pieces of consciousness atoms and let them be elements of a set. This set is the consciousness λ set.
- 029: The number of elements λ is restricted to an odd number more than 3 and less than p .
- 030: Consciousness λ set is being consecutively formed from its smaller occupying space. The occupying space is a total of spatial notions of logical atoms which belong to the consciousness λ set. This definition of an occupying space can be applied to any sort of set to be described later.
A significant space is defined to an occupying space. It refers to a spatial notion which is positioned in the middle of the size of all the spatial notions of consciousness atoms which belong to the consciousness λ set. Those spatial notions are aligned in an ascending order.

How to define a significant space is uniformed and consistent with the way of defining the occupying space of various types of sets.

- 031: If the number of the consciousness atoms is p , the number of the generated consciousness λ set is as follows: $C_p^3 + C_p^5 + \dots + C_p^\lambda$. C_p^λ is a sign of combination.
- 032: When consciousness λ sets are determined, any one of the permutations is chosen, which has those sets as its elements. This is denoted by consciousness λ column. Consciousness λ column is established not on the positioning locus, but in the spatial notion of IDS. Therefore, the consciousness λ column will remain even though the positioning line disappears.
- 033: According to the order of the consciousness λ sets aligning in the established consciousness λ column, a substance is chosen which can correspond to the consciousness λ set. Such an opting rule is called denotative unit. The denotative unit means to choose one consciousness atom of which spatial notion should be closest to the smaller occupying space of the consciousness λ set among from consciousness atoms which belong to the consciousness λ set. The chosen consciousness atom is copied also in IDS. The copied atom becomes an element of a different space from IDS. That different space is named consciousness space.
- Having a substance be formed, the consciousness λ set becomes attributes of the substance. The pairing relation between these attributes and the substance is called consciousness link. The consciousness link is used to describe the whole of existence. This means its substance is chosen with denotative unit. In the denotative unit, its attributes are considered to represent the whole.
- 034: The number of the admitted copying is restricted to the value of arrival of time velocity from UKS.
- 035: Substances of the consciousness λ set cannot be used in an overlapping way. This way of opting substances is a result from the operation of denotative unit.
- 036: Unless the substance is formed, the consciousness λ set becomes useless. In this case, the denotative unit works toward a consciousness λ set, which is placed next to the useless λ consciousness set. Thus, the greatest possible number of consciousness links is to be created.
- 037: Just like IDS's spatial notion, the consciousness space will not disappear even though the positioning locus disappears, so that the consciousness space will expand along with every formation of substance.
- 038: The consciousness link of which substance is a boundary atom is called critical link. As far as critical links are concerned, its substances used in critical links are allowed to be copied as much as possible.

4.2.2. Cognitive λ Set, Cognitive λ Column, Probabilistic Link, Probabilistic Space, Normalization Link, and Normalization Space

- 039: The greatest possible number of sets of which elements are cognitive atoms is made in IDS in the same way of producing consciousness λ set. Such a set is named consciousness λ set.
- 040: When the cognitive λ set is determined, a cognitive λ column is formed in a spatial notion of IDS which has all cognitive λ set as its elements. This formation is similar to that of cognitive λ column. However, the cognitive λ column is different from consciousness λ column in that it is a permutation of the cognitive λ sets. That is, it is arbitrarily chosen after all the possible permutations are formed.

- 041: A substance is chosen that can correspond to the cognitive λ set, which is placed at the top of the cognitive λ column. Such an opting rule of the substance is called connotative unit. The connotative unit means to choose one cognitive atom of which spatial notion should be closest to the largest possible occupying space of the cognitive λ set among from cognitive atoms which do not belong to the cognitive λ set. When the chosen cognitive atom is also copied, the reproduced one goes to another space to become an element of that space since it is not allowed to coexist in IDS. That space is called probabilistic space. That cognitive atom is called probabilistic atom. Once a probabilistic atom is determined, the cognitive λ set becomes attributes. The pairing relationship between the attributes and the substance is called probabilistic link.
- 042: Probabilistic atoms can be overlapped neither with each other nor with after-mentioned multiplex atoms. This is led by the operation of the connotative unit.
- 043: Unless the probabilistic atom is formed, the cognitive λ column becomes useless. Further development of the word of "providence" is not anticipated until a new time velocity comes out.
- 044: Just like the consciousness space, the probabilistic space will not disappear even though the positioning locus disappears so that the probabilistic space will expand along with the establishment of probabilistic atoms.
- 045: Once a probabilistic link is formed, the consciousness space chooses a substance of the consciousness link of which spatial notion should be closest and larger than the spatial notion of the probabilistic atom. This operation is called disclosure of the consciousness space.
- 046: From among the cognitive λ columns, a cognitive λ set is chosen, of which occupying space is larger than a spatial notion of the substance of the consciousness link. That cognitive λ set should be chosen with the same number of consciousness atoms which belong to the attributes of the consciousness link. This operation is called association of the consciousness space.
- 047: The substance of this cognitive λ set is chosen and copied according to rules of the connotative unit. A cognitive atom that is used as the substance is called normalization atom. Provided, however, that the normalization atom will be chosen not to be overlapped with already existing probabilistic and normalization atoms. This is led by the operation of the connotative unit. Once the normalization atom is determined, the cognitive λ set becomes attributes of this normalization atom. The pairing relationship between these attributes and the normalization atom is called normalization link. A space of which elements are normalization atoms is called normalization space. Just like the consciousness space, the normalization space will not disappear even though the positioning locus disappears, so that the normalization space will expand along with the generation of normalization atoms.
- 048: Unless the normalization atom is formed, further development of the work of "providence" is not anticipated until a new time velocity comes out in IDS.
- 049: When logical atoms are redefined due to the arrival of time velocity, the closest-placed consciousness atom to the boundary atom turns itself into a cognitive atom, otherwise the cognitive atoms becomes a consciousness atom. The former is called transposition, whereas the latter called reverse transposition.
- 050: If a boundary atom changes into a cognitive atom due to the work of transposition, the boundary atom disappears from the consciousness space. Likewise due to the work of reverse transposition, a cognitive atom which becomes a boundary atom disappears from

probabilistic, normalization, and after-mentioned multiplex and natural spaces. In this case, the number of allowable copying of consciousness and cognitive atoms which disappear is recovered with the same number of times.

- 051: In the same manner, the attributes of a link where boundary and cognitive atoms are mixed due to the work of transposition and reverse transposition is resolved. However, the number of allowed copying is not recovered.
- 052: If substances disappear, but their pairing attributes are not affected in transposition and reverse transposition, those attributes will not stay there.

4.2.3. Multiplex Link, Multiplex Space, Multiplex Coupling, Coupling Multiplex Link, Natural Link, and Natural Space

- 053: If the normalization link is produced and the number of cognitive atoms belonging to the attributes of the normalization link is r pieces, r pieces of cognitive atoms are taken from there to constitute sets, with some of them being overlapped. The number of the set to be made in IDS is $(r^r - 1)$. This set is called multiplex λ set.

Just like the case of cognitive λ set, cognitive atoms that become elements of the multiplex λ set do not newly come from UKS exclusively for this set. Therefore, this set forms some patterns on the positioning locus.

But one excluded set refers to a cognitive λ set that becomes attributes of a normalization link that causes the generation of this set.

- 054: If a substance is established in this multiplex λ set, this multiplex set becomes attributes. The substance is called multiplex atom.

The multiplex atom is not chosen with connotative and denotative units, and is formed by a copy of the normalization atom that brings the multiplex λ set into being. Therefore, a normalization atom and a multiplex atom are overlapped.

If the number of copying exceeds the allowable limit, the multiplex link is not established. In this case, a further development of the operation of "providence" is not anticipated until a new time velocity comes out in IDS.

A pairing relationship between a multiplex atom and its attributes is called multiplex link. All consciousness links are made at once based on the consciousness λ column.

As for the probabilistic link, only one of them is produced toward the cognitive λ column. In terms of the pairing relationship, it produces a normalization link. On the other hand, multiplex links are to be made toward one normalization link until the number of these links reaches $(r^r - 1)$. Cognitive atoms that belong to the attributes are overlapped unlike attributes of other links.

But multiplex atoms are transferred to a different space from IDS just like the cases of probabilistic and normalization atoms. A set of which elements are multiplex atoms is called multiplex space.

Just like consciousness space, probabilistic space, and normalization space, multiplex space will never disappear even though the positioning line disappears. Therefore, this space is expanding along with the formation of multiplex atoms.

- 055: After forming of the above-mentioned multiplex link, all the already existing multiplex atoms and after-mentioned coupling multiplex atoms (hereinafter simply called multiplex atoms) are aligned in an ascending order in terms of spatial notion. This is called multiplex column. The multiplex column is made one at a time in the multiplex space. Multiplex columns are accumulated in the consciousness λ column, and cognitive λ columns are accumulated in IDS. But multiplex columns are not accumulated in the multiplex space. Detailed explanation will be given in the latter section.

All possible combinations of two multiplex atoms are to be made on a basis of multiplex atoms placed in the multiplex column. This is called multiplex main-subordinate pair. Columns are made in the multiplex space, the columns which have multiplex main-subordinate pairs as its elements. This is called multiplex main-subordinate pair column. This will be given again detailed explanation.

Multiplex coupling is a name of another operation of the multiplex space which generates sets which have cognitive atoms as their elements, according to the placement order of the multiplex main-subordinate pair in the multiplex main-subordinate column. Therefore, this set makes some patterns described on the positioning locus just like multiplex λ set. It is called coupling multiplex λ set.

After a coupling multiplex λ set is made and becomes attributes, its substance is chosen according to the rules of connotative unit. This substance is called coupling multiplex atom. A pairing relationship between coupling multiplex coupling multiplex λ set and coupling multiplex atoms is a link, named coupling multiplex link.

Unless cognitive atoms produced with connotative unit are copied, the multiplex coupling becomes useless.

The operation of turning cognitive atoms into coupling multiplex atoms continues. When the repeated copying operation exceeds the allowed times, coupling multiplex atoms are not formed. In this case, multiplex coupling works toward its next-placed multiplex main-subordinate pair in the multiplex main-subordinate column.

Multiplex coupling is in progress, following the multiplex main-subordinate pair column which generate conditions of stop or end. As for the conditions, they will be discussed later.

Multiplex atoms and coupling multiplex atoms that are not incorporated into the multiplex column are taken up in the next multiplex columns when a new multiplex column is made. This is called renewal of the multiplex column.

- 056: Multiplex atoms and coupling multiplex atoms can be overlapped. Because of such a complex overlapping, the multiplex space is in chaos. Multiplex coupling is a natural operation to solve this chaos, but this disorder will never be placed in order unless that world is resolved.
- 057: There are some cases when the number of cognitive atoms belonging to the coupling multiplex λ set is even. When they become attributes, the nature of it links is different from multiplex link, called natural link. The substance of a natural link is called natural atom. Natural atoms form a set, that is called natural space.
- 058: After generation of natural links, this multiplex coupling ceases and a new coupling begins. This coupling is called natural coupling. Again its detailed explanation will be given.
- 059: When the number of cognitive atoms belonging to the attributes of a link which is generated with natural coupling is odd, this link is a coupling multiplex link. In this case, natural coupling ceases and then multiplex coupling begins.
- 060: When a time velocity comes in during the operations of multiplex coupling or natural coupling, the coupling operation ceases.
- 061: When the operation changes from natural coupling to multiplex coupling, multiplex main-subordinate column is renewed. The operation of multiplex coupling resumes from the main-subordinate pair which is placed after the previously ceased main-subordinate pair. Such a mechanism of the operation is also applied to the shift from multiplex coupling to natural coupling.
- 062: If natural links are no more created even though multiplex coupling proceeds until the last main-subordinate pair in the multiplex main-subordinate pair column, this multiplex

coupling finishes its operation and waits for an arrival of another time velocity. When a new time velocity comes out during the multiplex coupling, the multiplex coupling ends.

4.2.4. Natural Coupling, Coupling Natural Link, Singular Link, and Natural Space

063: After forming of the above-mentioned natural link in the multiplex coupling, all the already existing natural atoms and after-mentioned coupling natural atoms (hereinafter simply called natural atoms) are aligned in an ascending order in terms of spatial notion. This is called natural column.

Natural columns made in natural space are usually in plural number unlike multiplex column. But like multiplex columns, they will not be accumulated in the natural space. Details will be explained later.

Natural main-subordinate pair is formed with the same rules for the formation of multiplex main-subordinate pair. Natural main-subordinate pair column is also formed with the same rules for the formation of multiplex main-subordinate pair column. Detailed explanation on it will be given later again.

Multiplex coupling is a name of another operation of the multiplex space which generates sets which have cognitive atoms as their elements, according to the placement order of the multiplex main-subordinate pair in the multiplex main-subordinate column. Therefore, this set makes some patterns described on the positioning locus just like multiplex λ set. It is called coupling multiplex λ set.

After a coupling natural λ set is made and becomes attributes, its substance is chosen according to the rules of connotative unit. This substance is called coupling natural atom. A pairing relationship between coupling natural λ set and coupling natural atoms is a link, named coupling natural link.

If a needed cognitive atom does not exist, a boundary atom can be its substitute. As long as its copied atom exist, that becomes a coupling natural atom. Such a coupling natural link is called singular link.

A set of coupling natural links created from determined natural main-subordinate pair column is called history of coupling. A singular link is one of natural links which belong to the history of coupling. A natural link to be chosen for the establishment of the mechanism of consciousness is the one selected from the history of coupling to reach a singular link.

Natural coupling continues to work following the natural main-subordinate pair column until some conditions for stop or end are generated.

Coupling natural atoms that are not incorporated into the natural column are taken up in the next natural columns when a new multiplex column is made. This is called renewal of the natural column.

064: Natural atoms and coupling natural atoms can be overlapped. Because of such a complex overlapping, the natural space is in chaos. Natural coupling is a natural operation to solve this chaos, but this disorder will never be placed in order unless that world is resolved.

065: There are some cases when the number of cognitive atoms belonging to the coupling natural λ set is odd. When they become attributes, its link is multiplex link.

066: After multiplex links are generated with the natural coupling, the natural coupling ceases its operation and then multiplex coupling begins. This is considered as a condition for stop of natural coupling.

067: If multiplex links are no more created even though natural coupling proceeds until the last main-subordinate pair in the natural main-subordinate pair column, this natural

occupying space of its attribute.

Example 1: If the total number of multiplex atoms and coupling multiplex atoms is K , the following multiplex column is made if its spatial notion is denoted by S^{++} :

$$[S^{++}_1 < S^{++}_2 < \dots \leq S^{++}_i < S^{++}_j < S^{++}_k]$$

- 071: From the above-mentioned example of the multiplex column, $K(K+1)/2$ pieces of multiplex main-subordinate pair column are defined. When the multiplex main-subordinate pair is placed in a column, which is called multiplex main-subordinate pair column:

$$[S^{++}_1 | S^{++}_2]_1^1, [S^{++}_1 | S^{++}_3]_1^2, \dots, [S^{++}_1 | S^{++}_k]_1^{k-1}$$

$$[S^{++}_2 | S^{++}_3]_2^1, [S^{++}_2 | S^{++}_4]_2^2, \dots, [S^{++}_2 | S^{++}_k]_2^{k-1}$$

Likewise, the last line is to be $[S^{++}_{k-1} | S^{++}_k]_{k-1}^1$

- 072: Left-hand side of multiplex atoms of the multiplex main-subordinate pair is defined as main, whereas the right-hand side one is defined as subordinate.
- 073: Multiplex coupling law is an operation of creating coupling multiplex λ set that are attributes of the coupling multiplex link from multiplex main-subordinate pair. Details of the law is prescribed as follows:
- 1) Cognitive atoms that are attributes of main and subordinate of the multiplex main-subordinate pair are placed in an ascending order from smaller to larger in terms of spatial notion.
 - 2) Compare the spatial notions of cognitive atoms that are placed in the same position of the main and the subordinate of the column. Suppose the number of the cognitive atoms of the main be δ and the number of the cognitive atoms of the subordinate be μ .
 - (1) When $\delta \geq \mu$, that comparison of the spatial notions of cognitive atoms which are placed in the μ th and before is to be conducted. The comparison is not conducted for the cognitive atoms in the main which are placed in the order of $(\mu + 1)$ and after.
 - (2) When $\delta < \mu$, that comparison of the spatial notions of cognitive atoms which are placed in the δ th and before are to be conducted. The comparison is not conducted for the cognitive atoms in the subordinate which are placed in the order of $(\mu + 1)$ and after.

If the spatial notion of the cognitive atom in the main is larger, the cognitive atoms in the main and subordinate can be elements of the coupling multiplex λ set generated by this coupling.
 - 3) If the spatial notion of the cognitive atom in the main is larger, the cognitive atoms in the main and subordinate can be elements of the coupling multiplex λ set generated by this coupling.
 - 4) If the same cognitive atoms are found in this comparison, cognitive atoms that will later be determined are neglected.
- 074: If the coupling multiplex λ set is set, its substance is chosen with the connotative unit and its copy is formed, leading to the formation of a link. In this case, the number of cognitive atoms belonging to the coupling multiplex λ set is even, the link turns itself into a natural link.

4.3.2. Law of Natural Coupling

- 075: The way of determining a natural column is the same as that of a multiplex column. However, a difference lies in that the number of generated natural columns is not one, but several.

Example1: When the total number of natural atoms including coupling natural atoms is four, the following nine natural columns are created if their spatial notions are expressed with S^{++}_i :

- 1) $[S^{++}_1 < S^{++}_2 < S^{++}_3 < S^{++}_4]$
- 2) $[S^{++}_1 < S^{++}_2 < S^{++}_3]$
- 3) $[S^{++}_1 < S^{++}_2]$
- 4) $[S^{++}_1 < S^{++}_3]$
- 5) $[S^{++}_1 < S^{++}_4]$
- 6) $[S^{++}_2 < S^{++}_3 < S^{++}_4]$
- 7) $[S^{++}_2 < S^{++}_3]$
- 8) $[S^{++}_2 < S^{++}_4]$
- 9) $[S^{++}_3 < S^{++}_4]$

076: When natural columns are generated, the natural main-subordinate pair and a matrix of which elements are these natural main-subordinate pairs are defined for each of the natural columns. When the elements of this matrix are aligned in column, what we get is natural main-subordinate pair column. Let us show a case of 2) in the example 1.

$$\begin{aligned} & [S^{++}_1 | S^{++}_2]_1^1, [S^{++}_1 | S^{++}_3]_1^2, [S^{++}_1 | S^{++}_4]_1^3 \\ & [S^{++}_2 | S^{++}_1]_2^1, [S^{++}_2 | S^{++}_3]_2^2, [S^{++}_2 | S^{++}_4]_2^3 \\ & [S^{++}_3 | S^{++}_1]_3^1, [S^{++}_3 | S^{++}_2]_3^2, [S^{++}_3 | S^{++}_4]_3^3 \\ & [S^{++}_4 | S^{++}_1]_4^1, [S^{++}_4 | S^{++}_2]_4^2, [S^{++}_4 | S^{++}_3]_4^3 \end{aligned}$$

077: Left-hand side of the cognitive atoms of the natural main-subordinate pair is defined as main, whereas its right-hand side is defined as subordinate.

078: An operation of creating the coupling natural λ set from the natural main-subordinate pair is called natural coupling law. The operation of natural coupling law works the same way as that of the multiplex coupling law.

079: Natural coupling law works for natural main-subordinate pairs in the all natural main-subordinate pair column in a consecutive and parallel manner.

5. Three-dimension-like Space Model (TDM), Scenario Function (SF), Predicate Structure (PS), Proposition, Coordinates, Original Point, Words, and Complementary Work

080: The five mechanisms are reconstructed to be a model by the use of probabilistic, normalization, and multiplex links. This model is called the three dimension-like space model, abbreviated as TDM, shown in Figure 4.

081: The mechanisms of intention and consciousness can coexist in its space of TDM.

082: The probabilistic, normalization, and multiplex links have roles to form natural links to be objectified. They are to be re-described with a natural link where the mechanism of memory is built. This is an important operation to give a specific definition to TDM. Redefined links are called proposition or vector. There is no exaggeration to say that all the hypotheses in this study exist only for this purpose. The structure of re-described link is called predicate structure (PS), shown in Figure 5.

083: Propositions are placed on each coordinate of TDM, called probabilistic, normalization, and multiplex spaces. The mechanisms of intention and consciousness are to be described with propositions. In other words, TDM is considered to play a role to reverse the irreversible operation of providence by using propositions. The propositions are indicated as dotted lines in Figure 4.

084: An intrinsic nature of "word" is in common with an idea of a logical atom. Words 1) can be memorized; 2) are used to duplicate the structure of a link; 3) are used to

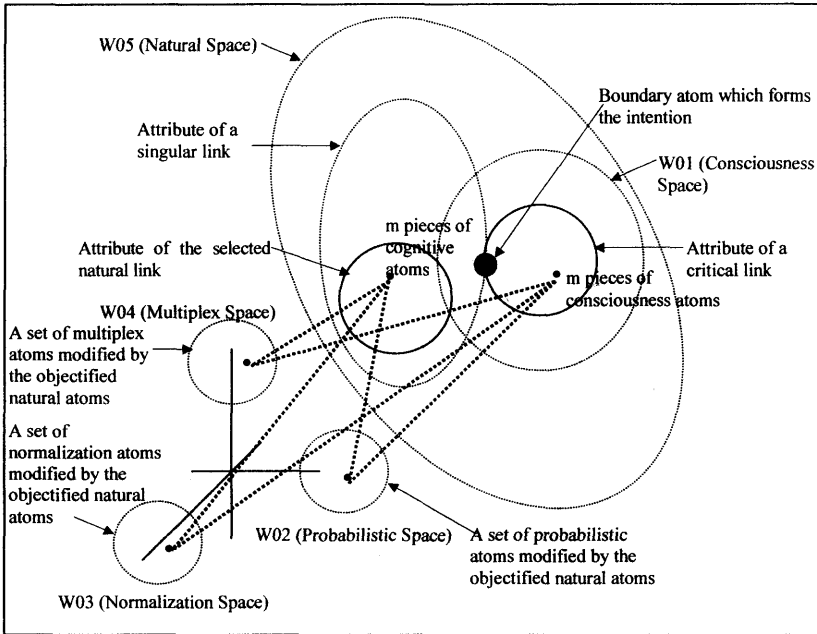


Figure 4. TDM and Propositions (expressed in bold and dotted lines)

duplicate the mechanism of coupling; and 4) are used to assume objectification. Thus, propositions can be defined with words.

085: TDM that is defined with function is called scenario function (SF).

086: When TDM that is expressed in a programming language is executed in computer, complementary work among propositions is conducted. Usually, we humans trace back our memories and make logic. Such a human work is replaced by the complementary work.

With the definition of proposition, complementary work is considered the same as the set up of the mechanism of memory. The mechanism of memory is built on a premise of the mechanisms of intention and consciousness so that even though we are not allowed to see their actual states, the mechanism of memory to be constructed means the formation of the mechanisms of intention and consciousness.

6. Various Concepts Used for Lyee's Software Development Methodology

Let us apply the axiomatic definition gained from the hypothetical world to the methodology of software development. Details on this matter were already explained in other papers and articles so that we intend here to focus on the relationship between those concepts and the hypothesis.

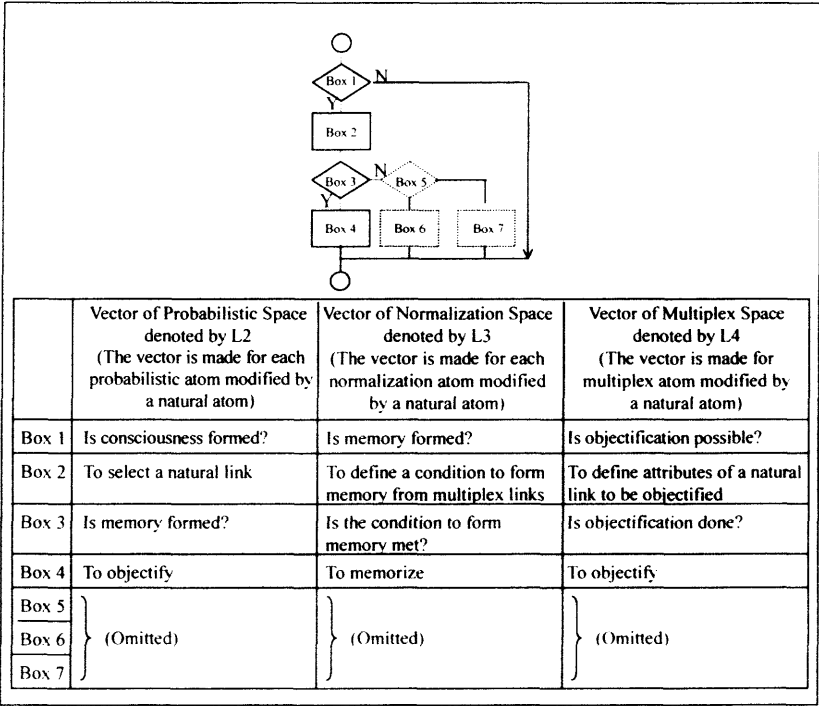


Figure 5. Structure and the Relations of Lyee Vectors (Intention, Consciousness and Memory)

- 087: TDM is regarded as a structure to govern the five mechanisms. A structure to govern the existence and the non-existence in a simultaneous way is defined with the five mechanisms that are formed with a singular link. TDM is that structure, that can be called synchronous structure.
- 088: If TDM is expressed in programming language, what we can get will be a Lyee's program.
- 089: A set of TDM is also considered as TDM, but it is given another name, process route diagram abbreviated as PRD when it is used in software development methodology.
- 090: Probabilistic, normalization, and multiplex spaces that take the roles of axes of coordinates of TDM are replaced by three types of pallets. Each pallet of these three is denoted by W02, W03, and W04. Consciousness and natural spaces that are put in a space of TDM are called W01 and W05 respectively.
- 091: W02 contains programming elements such as 1) descriptive propositions for probabilistic space (L2); 2) propositions for input action (I2); and 3) routing propositions for probabilistic space. It is defined by a controlling program of probabilistic space (Φ2).
- 092: W03 contains programming elements such as 1) descriptive propositions for normalization space (L3); 2) routing propositions for normalization space (R3). It is defined by a controlling program of normalization space (Φ3).
- 093: W04 contains programming elements such as 1) descriptive propositions for multiplex space (L4); 2) propositions for output action (O4); 3) propositions for structural action

- (S4); and 4) routing propositions for multiplex space (R4). It is defined by a controlling program of multiplex space (Φ_4).
- 094: The structure of all propositions is represented by PS and generically called vector. In Lyee's theory, an operation of the natural world is hypothesized as coupling. This operation in TDM can be substituted by a computer.
- 095: Φ_2 , Φ_3 , and Φ_4 are the programs to let all propositions on their respective pallets work and take each of their roles as their respective spaces. The structures of Φ_2 , Φ_3 , and Φ_4 are not prescribed by PS. But their logical structure is deterministically given.
- 096: There is one program necessary to govern these three spaces in order to create a synchronous state in those spaces. That program is called tense control function denoted by Φ_0 (12). Φ_0 perform its duty in an associative work with the operations of (Φ_2 , Φ_3 , Φ_4) and (R4, R2, R3). The structure of Φ_0 is not prescribed with PS. And again its logical structure is deterministically given. However, let us give one notice that it is advisable to define those programs in programming language, considering the executing capability of computers.
- 097: A structure of Φ_0 in PRD can be formulated in a similar way to that of TDM.
- 098: The above-mentioned R3 (6) needs programming elements such as R3R, R3C, R3D, and R3M. R3R is placed for a recursive control of TDM. R3C is placed to form synchronicity with W04 of TDM that is positioned right after the concerned SF on PRD. R3D is placed to form synchronicity with W03 of TDM that is positioned just before the concerned SF on PRD. R3M is placed to form synchronicity with W04 of TDM that is placed a few SFs ahead of the concerned SF.
- 099: L4 and L3 are defined for every output word that is used for O4.
- 100: L2 is defined for every input word that is used for I2.
- 101: Dotted lines in Figure 4 show L4, L2, and L3.
- 102: Words are largely divided into two: one is user's word or regular word; the other is called k-word, the one that systems engineers add for supplement.
- 103: In terms of nature, words are also classified into several categories such as input word, output word, equivalent word, boundary word, empty word, summation word, given word, and self word.
- 104: S4 is a proposition to maintain a synchronous nature among each unit of internal memory area so that the number of S4 to be made is no more than the number of internal memory areas (its minimum unit is a word).
- 105: I2 and O4 are needed with the same number of types of logical unit of DB and so forth.
- 106: L4, L2, L3, R4, R2, R3, and S4 are propositions that govern the synchronicity of TDM. I2 is a proposition which governs synchronicity between TDM and external memory device. O4 is a proposition which governs synchronicity between TDM and external memory device.
- 107: PRD is defined with pallets (W04, W02, W03) and routing action propositions (R4, R2, R3). This diagram is named process route diagram.

7. Conclusion

Our daily life, being affected by personal interpretation of the phenomena, shows ever-changing aspects like kaleidoscope. The fundamental world which we cannot comprehend, however, seems to be tranquil and universal with little change.

Lyee methodology is a method to capture software by using an idea of the demarcation between the changing world (the part) and the unchanging world (the whole). All the necessary

concepts to make this idea work are derived from the hypothesis. Thus, the Lyee method may not always be subject to the engineering-oriented way of thinking.

With this method, however, it becomes possible to clearly define ambiguous concepts. For example, a minimum unit of software, called predicate structure or PS, can be determined with the universally valid structure. Those minimum units which are elements of a set constitute software. Such an idea is indeed contributing to overcoming various problems of software.

Since this method has already been used for various system developments, its entire process can be openly evaluated. Up until now, people use a conventional way of thinking to make logic when they handle requirements and make software programs. The Lyee's method, however, is relieved from inevitable obstacles caused by such a logic-building method.

In other words, the Lyee's method claims that the work of building logic for the system development, which is implicitly commissioned to the persons involved, can be replaced by a set of PS and computers, thereby achieving dramatically increased efficiency.

8. Acknowledgement

I would like to apologize for my own immaturity to the existence of nothing, although my awareness of it leads me to conduct the research into this study. I feel still deeply moved by Professor P.R. Masani, who was already deceased, suggested that the existence of nothing should be a clue to look over profoundness of an issue of the existence and give solutions to software problems, and encouraged me by pointing out the importance of this kind of study. I sincerely wish to pay the greatest respect to him. With a constrain of the paper, I dare not to mention names of my devoted staffs, but my thanks go to them as well as my beloved talkative cats. Last, but not the least, I wish to extend my sincerest thanks to Mr. Yoshitsugu Komiya, president of Catena Corporation, who provides greatest support to me, and to Professor Hamido Fujita of Iwate Prefectural University who gives me academic advice.

References

- [1] F. Negoro, Principle of Lyee Software, Proceedings of 2000 International Conference on Information Society in the 21st Century (IS2000), pp. 441-446. 2000.
- [2] F. Negoro, Intent Operationalisation for Source Code Generation, Proceedings of the World Multiconference on Systemics, Cybernetics and Informatics (SCI2001), Volume XIV, Computer Science and Engineering: Part II, pp. 496-503. 2001.
- [3] I. A. Hamid, *et al.* New Innovation on Software Implementation Methodology for 21st Century, Proceedings of the World Multiconference on Systemics, Cybernetics and Informatics (SCI2001), Volume XIV, Computer Science and Engineering: Part II, pp. 487-489. 2001.
- [4] F. Negoro, *et al.* A Proposal for Intention Engineering, Proceedings of the International Conference on Advances in Infrastructure for Electronic Business, Science and Education on the Internet (SSGRR2001), CD-ROM. 2001.
- [5] F. Negoro, The Predicate Structure to Represent the Intention for Software, Proceedings of the ACIS 2nd International Conference on Software Engineering, Artificial Intelligence, Networking & Parallel/Distributed Computing (SNPD'01), pp. 985-992. 2001.
- [6] F. Negoro, A Proposal for Requirement Engineering, Proceedings of the 5th East-European Conference on Advances in Databases and Information Systems (ADBIS2001), Volume 2, Tutorials, Professional Communications and Reports, pp. 7-18. 2001.
- [7] F. Negoro, Method to Determine Software in a Deterministic Manner, Proceedings of the International Conferences on Info-tech & Info-net (ICI2001), Conference D, pp.124-129. 2001.

Chapter 1

Intention and Hypothetical Oriented Software

This page intentionally left blank

Intention as Architecture of Spaces

Liliana ALBERTAZZI

*Department of Sociology and Social Research, Trento University
Via Verdi 26, 38100 Trento Italy*

One of the theoretical presuppositions of Lyee methodology is the concept of *intention*, which rules the search for information by the user. Specifically, one of the main Lyee's meta-principles affirms that *only the user knows what he/she intends* (the so called intentionality thesis). The intention presupposition is also what specifically characterises and distinguishes Lyee methodology from the others methodologies on the market.

This starting point establishes a direct connection between the source of information and cognition, as the search for information and its acquisition depends, at least partially, on the user's viewpoint, which becomes a part of the process itself.

What is required of a methodology so that it can flexibly adapt to the individual user, also in consideration of the fact that the user's intention, while operating, can change direction and interest, according to the diverse inputs encountered?

The challenge raised by Lyee for the computer science society is really daring.

Since one of Lyee main goals is to construct a platform able to automatically generate almost all the codes necessary for information, from a conceptual and a methodological point of view the questions that Lyee must answer are the following:

1. Is this point of view, as a subjective cognitive scheme, structurally constitutive of information?
2. Are other cognitive schemes able to account structurally for how information is generated?
3. What is the relation between cognitive schemes and the nature of intention to which Lyee refers?
4. What is the relation among sources of information, cognitive subjective structures and the words used by the program?

1. Lyee's underlying theses

As to the source of information, in his papers Negoro refers to a concept of reality viewed as unknowable continuous flux of ever-changing forces and configurations (substrate space) ([12], [13], [14]. See also [15]). In principle, then, a Heraclitean space of this kind comprises every kind of stuff, including sensitivity, spirit, consciousness, motor activities and emotions, with both a physical and an experiential base. Intentions directed to get information out of the substrate space are then the introduction of some discontinuities within the basic domain.

From this philosophical assumption several consequences follow for Lyee methodology:

1. Given the incommensurability of such a substrate space, the only possible operation with which to acquire information is that of intentionally reducing its complexity (density) individuating and identifying within it partial 'entities' of various kind than can be manipulated as atoms of information. It follows that intentions – as selection operations – are constitutive of the individuation principle of the entities about which we desire to obtain information.

2. Also it follows, in principle, from the nature of the substrate space, that the entities selection operation can be iterated indefinitely: new intentions, in fact, continuously select and individuate new entities, as multiple and multifarious aspects derived from the original whole.

In Lyee methodology, the entities individuated through intentions are called *objects* or static items (products of the intention operations), which interact dynamically according to the sequence of the process route vectors.

The objects are represented by a three-dimensional modelling space consisting of:

- (i) The information coming from the environment external to the object (W02)
- (ii) The object's internal structure (i.e. how information is conceptualised from a specific point of view) (W03)
- (iii) The information conveyed from the object (conceptualised information) to the outside environment (W04).

This implies that in Lyee the conceptual categories defining objects are not of the linguistic-definitional type but are based on inputs received from the physical world via the diverse sensory apparatuses.

For all these reasons the underlying Lyee ontology is intrinsically cognitive because there is a continuous interaction between the source of information (substrate space) and its individuation in entities of some kinds, through the operation performed by the user. In particular, in Lyee the focus of attention is the *actual* search for information performed by the user.

There are several aspects of the presentation of the principles and of the meta-principles underlying Lyee methodology which, however, may give rise to interpretative ambiguities if they are not adequately clarified. One of them is the synonymous use of 'intentions', 'objects' and 'words', which reveals an ambiguous relation among substance (source of information), cognitive operations (conceptual schemes for the individuation of entities), the products of operations (objects in a broad sense) and their semantic treatment (words).

A second aspect concerns the structure of intentions itself: are they eminently subjective points of view, or do they refer to constitutive structures of information, i.e. do they have also an ontological commitment?

A third aspect concerns the level-structure of reality. In Lyee methodology, in fact, the relation among a metaphysical base level (substrate space), a physical level of stimuli (which seems to coincide with the former), a cognitive level of individuation of the entities and a modelisation level of the whole procedure is not always clear.

When Mr. Negoro[12][13] speaks of the object in terms of 'substance covered by qualities', in fact, since substance in itself is analytically unknowable and is only a way to connect the object's qualities together, there is a sort of implosion of levels that produces interpretative ambiguities.

A further aspect concerns the nature of the object itself. On one hand this seems to be a dependent and derived part of the substrate space of eminently static nature, while on the other it is modified by the cognitive structures and modifies in its turn the subsequent

individuation and apprehension of other entities, thus acquiring a substantially dynamic role. In other words, the theoretical nucleus of Lyee rotates around the poorly specified concept of *object as product of intention*.

I shall consider some problems deriving from the above definition, trying to eliminate some of its interpretative ambiguities. I start from the concept of object as both a cognitive and a semantic product. These are two aspects that are particularly concerned with some characteristics of Lyee's three-dimensional modelling space. Here I shall leave apart the metaphysical implication of the concept of intention, which may be of less interest to this audience.

2. The concept of information in Lyee

As a consequence of Lyee assumptions, information is neither intrinsically semantic nor does it express a direct relation between states of the world and propositions (the Tractarianism hypothesis dominant in the twentieth century): in fact, between the substrate space and the information as conceptualised in objects there exists a complex procedure of cutting off and contextualising the substrate space by *intentions*.

In particular, semantic structures seem to be characterised in relation to domains and cognitive schemes that construe a certain situation in a specific way, from time to time imposing (through intention) a profile (object) on a base (background) on the substrate space. Semantic construal consists in the ability to conceptualise and cut off diverse *objects* according to different levels of specificity, focus, background, perspective and salience.

From a semantic point of view an object, as a product of intention, is not so much a word as a *lexical item* which evokes a series of cognitive domains which underlie its meaning. This meaning has both a prototypical nucleus and a structural peripheral semantic ambiguity. Consider the following example.

The object-word *glass* refers to a conceptual prototypical space which implies a certain form (cylindrical), a typical orientation (vertical axis), a function (containing liquids), a stuff (glass), a certain standard size, etc. At the same time the word *glass* implies a series of similar conceptual domains which can concern, from time to time, its being an artefact, its cost, its design, the place it usually has on the table, etc., involving other conceptual spaces at different hierarchical levels. In other words, the lexical item *glass* *implicitly* comprises all the aspects related to its pertinent domains, which from time to time can be activated and made explicit by intentions.

Consequently the structure of object-word is made up of conceptual spaces progressively embedded in it implies a structural and intrinsic polysemy. This, however, does not mean fuzziness, since it is ruled by the invariants that are transposed through the various connected domains. That polysemy and ambiguity are characteristics of the intentions expressed by the user is also a basic meta-principle of Lyee, which seeks to compensate for its irreparableness through the continuous iteration of its basic structure until a situation of overall stability is attained.

Let us consider another object-word, namely *ring*. Its prototypical meaning of 'circular entity' implies spaces of conceptualisation ranging from jewellery (then on turn from social institutions, custom (wedding ring), etc.), to astronomy (Saturn), to sport (the rings), to botany, to zoology (annelid), to shipping (mooring rings), etc.

This shows that knowledge is not organised in atomic separated parts, as Lyee maintains. Rather, it is organised into holistic structures that bring together diverse information entities (or parts of them) in Gestalt wholes (Fillmore's concept of frames, scripts, etc. in [5]). In searching for and producing information, Lyee methodology must take into account an encyclopaedic complexity, which shows no precise boundary between linguistic and non-linguistic knowledge. A lexical item, an object-word as product of

intention is an access route to a set of conceptual domains, of which some are central (prototypical), other peripheral; all of them, however, are implicitly contained in its specific conceptual space. From this point of view, Lyee cannot address a truth conditional semantics, or define object-words in terms of necessary and sufficient conditions that would exhaustively determine the meaning of a word (as, for example, in [6]). The semantics of Lyee, in fact, must deal with the construction of a meaning that has strong contextual, dynamic, metaphorical valences due to the subject activity: these valences correspond to the continuous variation of the information proceeding from the substrate space and objectified in connected cognitive spaces and *vice versa*.

3. The structure of the correlation

The widely contextual nature of the information implies structural polysemy, but not an absence of structure. Starting from the substrate space, in fact, the information undergoes a series of construal operations in intentions, which are ruled by constraints that have at their basis some cognitive schemes the structure of which can be analysed.

Cognitive linguistics, for example ([1], [8], [10], [17]), has identified some basic schemes of conceptualisation, which are the following:

POINT OF VIEW
PATH
FIGURE/GROUND
WINDOWING OF ATTENTION
SCANNING
FORCE DYNAMICS

The nature of these cognitive schemes embedded in natural language confirms the Lyee hypothesis of a continuous route of information proceeding from the environment external to the object (as conceptualised information from a specific point of view), and then from the object to the external environment. Specifically:

The POINT OF VIEW or vantage point is the spot chosen for consideration of a certain scenario. Figure/ground organisation consists in a discontinuity located in the basic substrate space of information. The immediate consequence of adopting a specific viewpoint, in fact (in vision and in conceptualisation as well) is the structuring of the scene into a pattern determined by the configuration of an object on a background (see below), the correlated position of the subject determining the orientation, and the directions that regulate the orientation of the scene itself, or in other words, the alignment of the scene with respect to the axes of the visual field (mainly horizontal and vertical) ([9], pp. 262-7). Briefly, the point of view is a vantage point from which the information is viewed, due to the subject's orientation, the axis of direction along which the scene is oriented, the subjective-objective opposition in the construction of the scene, etc. Its result is the analysis of the object in its different possible perspective *aspects*.



FIG. 1.

The PATH SCHEME (FIG. 1.) is active in any form of conceptualisation and concerns that aspect of information connected to its 'beginning' and 'end'. Details of the path (whether linear, deviating, with gaps, etc.) are not included: they require lexical information to be further specified.

FIGURE-GROUND SCHEME is perhaps the most important one, and a typical Gestalt scheme. In fact, in perception as well as in conceptualisation, the role of figure is played by the object, which is viewed as a 'thing' (usually placed in front of the background), is smaller, is moveable (also in conceptual sense), and has contours. On the other hand, the ground, which extends behind the figure, has no form and appears to be more distant, etc. [16]. Obviously a figure can be the ground of another figure, for example two objects can be related to each other. The FIGURE-GROUND organisation represents the construal of an entity as central in comparison to another entity functioning as a reference point, i.e., as an entity with sufficient cognitive salience to be invoked *in order to providing cognitive access* to other associates entities.

FIGURE-GROUND scheme (FIG. 2. – 5.) has an internal structure consisting of four relations: *inclusion* ('The glass is in the cupboard'), *proximity* ('The bicycle is near the house'), *coincidence* ('The cork is on the bottle'), *separation* ('The gate is far from the house' or 'The house is far from the gate') (in the last case both objects are figures).

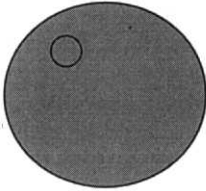


FIG. 2.

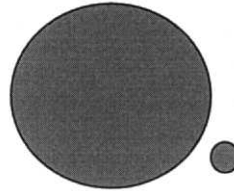


FIG. 3.



FIG. 4.



FIG. 5.



The FIGURE-GROUND scheme permits two fundamental cognitive operations: taking the *interconnections* among a set of indices as the base and *profiling the region* that derives from them; or using a *set of indices* as the base and *profiling their interconnections*. These two strategies give rise, for example, to the formation of nouns and verbs [10].

The WINDOWING OF ATTENTION SCHEME includes an assumed vantage point, whether the viewer is static or moving, and whether the situation is being viewed in objective terms or as the viewer experiences it. Moreover the viewing focuses on concepts to different degrees in different contexts, which gives rise to different levels of local information at different granularity. This scheme comprises *complex conceptual figures of attention* like the level, centre, range, focus of attention ([10], pp. 278-288).

The SCANNING SCHEME (FIG. 6.) comprises a series of conceptualisations concerned with *movement*, which is understood both as movement in physical space and a qualitative change or movement in a conceptual space. Rather than base/profile, *trajectory* (as primary focal participants in a profiled relationship) and *landmarks* play a key role here. There are two different types of scanning: *sequential*, where the focus is on the progressive change of one situation into another (e.g. 'falling'), and *additive*, where all events are considered as coexistent and simultaneous (e.g. 'fall'). Scanning also concerns the type of mental procedure and the point of departure characteristics of analysing a particular situation ([10], pp. 68-9).

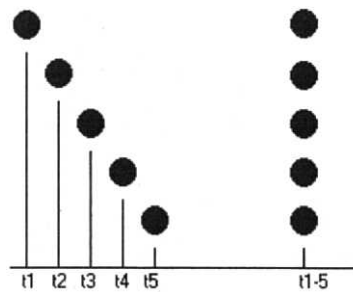


FIG. 6.

The FORCE DYNAMICS scheme concerns the forces exerted by the elements of the scene on each other; these forces may take the form of attraction, repulsion, fusion, inhibition, etc. The primary semantic roles of this scheme are those of 'agonist' and 'antagonist'. With this scheme it is possible to account for the *causative* dimensions and

origin of *modalities*, which constitute a closed class specified by force concepts. Obviously, this class comprises not only physical forces but also psychological, social, and interpersonal dynamics [17]. Example of such cognitive scheme, in fact, are to be found at different levels of conceptualisation as in the following examples: 'The ball kept rolling despite the stiff grass', 'Carla's father prohibits her to go to the park', 'Lily refrained from speaking' (in the last case the roles of agonist and antagonist are played by the same entity), etc.

It seems, then, that the modes in which information is constructed and expressed in language derives from corporeal schemes projected in abstract domains (as it happens, for example, in the cases of in/out, toward/from, high/low, verticality/horizontality, contact, support, etc.), which are kinds of topological and orientation structures active in building concepts inherent to spatial relations.

Lakoff[8] defines these structures as *image-schemes*, listing among them 'container' (in/out), 'part-whole', 'link', 'source-path-goal' (projection from a source domain to a goal domain), 'up/down' (verticality/horizontality), 'front/back', 'linear order', 'centre/periphery', etc. Each of these image-schemes keeps some structural characteristics of information: for example, the container scheme consists essentially in a boundary distinguishing an interior and an exterior [7].

These schemes govern the natural language because — starting from the substrate space, as Lyee would say — they are direct projections of the conceptualised information in the semantic structures.

A preliminary analysis of some of the cognitive schemes embedded in natural language shows as the intention structures the objects of information through a continuous mapping among diverse conceptual domains, starting from basic operations, which may be triggered by similarity of aspects, common sharing of conceptual frames, etc.

The examples mentioned above of object-words like *glass* or *ring* evidence the fact that the objects of information are the output of thousands of conceptual mappings (intentions), though ruled by constraints, which permits inferences among domains. This is the sense of the essential metaphorical character of natural language, intended not as arbitrary meaning but as general principles of mapping among diverse conceptual domains that characterises epistemic correspondences [8].

The concept of metaphorical mapping is essential to a methodology like Lyee because it decides the *use* of the language pertaining to the source domain and the inferential models for the goal domains (see the above mentioned examples of the lexical item *ring* as 'circular entity' and its correlated domains in zoology, custom, astronomy, etc.). In this way, intention establishes a correlation between two or more representations and can also establish new conceptual relations (in which lies the creative role of conceptual mapping among diverse domains). Such operations performed by the user's intentions are a common praxis in the search for and production of information, proceeding from the basic substrate space to cognitive and semantic ones.

As to the metaphorical nature of conceptualisation space, over time some image schemes are stored in natural languages, giving rise to a sort of *long term memory storage of information* specific of a certain culture. In some Western languages, for example, the metaphor 'time is money' underlies a series of common usage expressions like 'this work costs me one hour', 'you are making me loose time', etc.

Even more important, however, are the dynamics of cognitive spaces as a structure of temporary and partial organisation built by one who *actually* speaks, searches or produces information through operations of imagery projection from one domain to another, and through the subjective completion and elaboration of data of the source domain (Fauconnier's concept of 'blending' in [4]).

Since the cognitive spaces individuating the objects of information *simultaneously* represent a plurality of models and their relations, they are intrinsically *dynamic* spaces of modelling. Cognitive spaces, in fact, may also simultaneously concern diverse parts of information, like for example perceptual information, fictional or hypothetical states of affairs, past situations and abstract domains.

A conceptual system embedded in a natural language, then, has thousands of similar projections at various levels of conceptualisation, which a methodology like Lyee must be able to organise if its assumptions are valid.

4. Complete and incomplete objects

On the basis of the foregoing discussion we may conclude that the objects of information known through cognitive schemes are constitutionally *fragments of the substrate space*, which *necessarily* imply some form of completion by the intention.

From this point of view, all objects are in principle *incomplete objects* which require forms of cognitive intervention (intentions) to be realized in a particular form (this assumption, for example, underpins Meinong's ontology in [11]).

There are moreover different degrees of completion which increase in complexity according to the type of object considered. Some objects, in fact, permit more limited forms of completion, examples being those relative to sensory perception, where they are mainly articulations of bottom-up type; other objects have forms of completion which are in principle indefinite, and here it is mainly cognitive mechanisms of top-down type that operate.

As a matter of fact, the objects of sensory perception like balls, chairs, trees or bicycles are never given in all their sensory dimensions or in all their aspects. Moreover, there also objects that are *structurally incomplete*, like fictitious objects, scientific theories, works of art, apparent movements, organisms, and so on. In particular, there are objects that are essentially temporal (events and processes) like a thunderstorm, a battle, the fall of an apple from a tree, a melody, and objects that are only marginally such (chairs, bicycles, houses, mountains, etc.). This aspect in particular introduces a problematic element into any kind of software methodology that has ontological claims.

What, in fact, are the various roles performed by the intention in the cases of a stroboscopic movement, of a snowfall, of listening to a sonata, or when seeing a town square, a glass, a street? In particular, what roles are played by the above-discussed cognitive schemes in the construction of these various objects? And what is it that in each case holds together the different aspects of the substrate space considered? These questions are crucial for the Lyee ontology, which according to its assumptions (the substrate space) should be able to handle all the aspects of this complexity.

As a consequence of the fact that objects are given only in a *multiplicity of aspects* held together by the operation of intention, which takes place through cognitive schemes that map onto diverse cognitive domains, it follows that the objects in the program can be located and retrieved in a number of places. The lexical item *glass*, for example, as a complex category with a variety of interconnected senses, can be classified in diverse domains. Moreover, there are object-words that change nature according to the linguistic context (for example, 'candidate' from a person dressed in white) or which are intrinsically changeable (evolving objects). In this case, the only way out of the impasse for Lyee is to adopt a kind of reasoning that is as natural as possible and takes account of both the modes of construction and completion of objects (i.e. the schemes discussed above) and the type of object in consideration.

Once again, therefore, the role and the nature of intention is at the basis of the complexity and the difficulty.

5. Conclusions

In the opening I recalled that one of the theoretical assumptions of the Lyee methodology is the concept of *intention* or the so called *intentionality thesis* (for a possible integrating philosophical view see [3]). I also underlined that the assumption of intention characterises and distinguishes the Lyee methodology from the others on the market, hence its importance.

In this article I analysed only one side of the question i.e. that concerning the role of word-objects in Lyee, related to the problem of the search for information, showing the intrinsic cognitive and subjective point of view embedded in the words we use. There would be much more to say concerning the role of intentions as cognitive *operations* as such, independently at least partially of their *products* (objects and words), and also their objects before their being expressed in words.

This enquiry, however, would bring us on one side to the boundary of metaphysic so cherished by Negoro, on the other to the analysis of cognitive processes as such, and to the specific nature of cognitive space, both arguments requiring detailed analyses given their complexity.

As to metaphysics, in fact, one should be able to show how intention is the fundamental trait as a primitive operation of dynamic change in reality, and as to cognitive space one should be able to show its related primitives as direction, velocity, transfer of invariant features, etc. in producing information (a first approach in [2]). Both tasks deserve a further specific enquiry, parallel to the development of Lyee on effectual basis.

Acknowledgment

This paper hereof is contributed to the Lyee International Collaborative Research Project sponsored by Catena Corp. and the Institute of Computer Based Software Methodology and Technology.

References

- [1] Albertazzi, L. ed. 2000. *Meaning and cognition. A multidisciplinary approach*. Amsterdam: Benjamins.
- [2] Albertazzi, L. ed. 2002. *Unfolding perceptual continua*. Amsterdam: Benjamins.
- [3] Brentano, F. 1995. *Psychology from an empirical standpoint*. London: Routledge.
- [4] Fauconnier, G. 1994. *Mental spaces*. Cambridge: Cambridge University Press.
- [5] Fillmore, Ch. 1977. 'Scenes-and-Frames semantics', in *Linguistic structures processing*, ed. by A. Zampolli, 55-81. Amsterdam: North Holland.
- [6] Jackendoff, 1983. *Semantics and cognition*. Cambridge, Mass.: MIT Press.
- [7] Johnson, M. 1987. *The body in the mind: The bodily basis of meaning, imagination and reason*. Chicago, Ill.: University of Chicago Press.
- [8] Lakoff, G. 1987. *Women, fire, and dangerous things: What categories reveal about the mind*. Chicago, Ill.: University of Chicago Press.
- [9] Langacker, R. 1987. *Foundations of cognitive grammar. Vol I. Theoretical prerequisites*. Stanford: Stanford University Press.
- [10] Langacker, R. 1991. *Concept, image and symbol: The cognitive basis of grammar*. Berlin: Mouton De Gruyter.
- [11] Meinong, A. 1983 (1910). *On assumptions*. Berkeley: University of California Press.
- [12] Negoro, F. 2001a. 'A proposal for requirements engineering'. *Proceedings of ADBIS*. Vilnius, Lithuania.
- [13] Negoro, F. 2001b. 'Methodology to determine software in a deterministic manner'. *Proceeding of ICII*. Beijing, China.
- [14] Negoro, F. and Hamid, I. 2001c. 'A proposal for intention engineering'. *Proceedings SSRRR*. Rome.
- [15] Poli, R. 2002. 'Requirement shifters', forthcoming.
- [16] Rubin, E. 1921. *Visuell wahrgenommene Figuren. Studien in psychologischer Analyse*. Copenhagen: Gyldenalske.
- [17] Talmy, L. 2000. *Toward a cognitive semantics*, 2 vols., Cambridge Mass.: MIT Press.

Requirement shifters

Roberto Poli

University of Trento and Mitteleuropa Foundation, Bolzano

Abstract. Requirement shifters are context-dependent operators. The basic idea is to consider objects as items which, as regards at least some of their components, are functionally dependent on the perspective from which they are viewed. There may therefore be a plurality of requirements relative to the same object, with the corresponding possibility that the changes of requirements may be made to depend on the use of appropriate operators. Four different types of requirements shifters operators are distinguished (grouping, round-going, approaching and distancing, and base-highlighting). The long term idea is to enlarge the basic Lyee methodology by adding the family of requirement shifters operators. The actual paper presents a first purely conceptual analysis of the requirement shifters operators.

1. Introduction

One of the main meta-principles adopted by Lyee is that the user alone knows what s/he intends [1]. According to Lyee methodology, the final user alone decides for the correct organization of words (Lyee's basic units) and of their qualifications. This methodology may nevertheless run into serious troubles as soon as (1) the user himself do not know which decisions are the best for his/her own tasks; (2) there are many users with potentially or actually conflicting requirements (think about IR from the net).

If Lyee aims at being a really general methodology for software construction, regardless of the specificity of any domain application, it should be enlarged so that the above cases 1-2 can be dealt with.

It may therefore be reasonable to maintain the actual Lyee methodology as a *core methodology for default situations*, those where the final user is available and knows what he wants, and to add a series of new modules dealing with non default situations, as those listed by the two points above.

This paper presents a first purely conceptual analysis of some of the possibly many "requirement shifters" that may arise in dealing with non default situations.

The paper's title refers to the problem of contextual dependence, and therefore to the problem of the object as an item which, as regards at least some of its components, is functionally dependent on the perspective from which it is viewed. In a situation of this kind it is obvious that there may be a plurality of requirements relative to the same object, with the corresponding possibility that their change may be made to depend on, or at any rate be connected with, the use of appropriate operators. The paper shall analyze both *operators* which alter a requirement (the 'shifters' in the title) and *operations* which alter a requirement. Although the difference is a subtle one, it is important nonetheless: as we shall see, there are cases in which operations do not have a corresponding operator. The cases in question are essentially of two different types.

The first comprises situations in which there is no operator because the operation in question is only apparently *one* operation, whereas in reality it resolves into a *multiplicity* of different operations, eventually at diverse levels. In these cases, therefore, one should speak, not so much of an operator, as of a set of operators which jointly yield the result sought.

Secondly, there may not be an operator corresponding to an operation because the operation may be embedded in the structure of the reference field. In this case, the object may change in consequence of a change in the field. Here, therefore, one should speak of 'field modifiers' rather than of 'object modifiers'.

The paper is conceptually divided into two sections. The first is devoted to presentation of various requirement shifters. The second section connects a number of aspects that emerge from analysis of the requirement shifters with some more general issues.

We shall see in particular that at least some of the cognitive structures discussed require an ontological *anchoring* if they are to function properly. If they did not possess this ontological anchoring, it would be impossible to explain how they operate. Note the unusual terminology used. I have not spoken, in fact, of ontological *foundation* but of ontological *anchoring*, by which I mean a weaker form of connection than the traditional concept of foundation.

I distinguish four different types of requirement shifters, which I call (1) grouping shifters, (2) round-going shifters, (3) shifters of approaching and distancing, and (4) base-highlighting shifters.

I shall provide an outline of the first three types, concentrating in more detail on the fourth.

2. Grouping

The first category of cognitive operations comprises those which alter the perspective through processes which profile a figure with respect to a ground.

From his *Philosophie der Arithmetik* (1890) onwards, Husserl distinguished at least four different forms of grouping:

1. the *kollektive Verbindung*, or the grouping produced by the thinking together of various elements;
2. relations of spatial and temporal juxtaposition like contiguity and succession;
3. the *figurale Momente*, or those types of individual relations which connect geese into flocks, trees into avenues, dots into patterns, and so on;
4. the dependence relations between mutually correlated contents, as between colour and space ([2: 35], with inversion of the 3rd and 4th cases).

These four types of grouping display different levels of complexity: the third and fourth seem distinctly more complex than the first and second.

The types of grouping listed above display the following characteristic. As one passes from the first to the fourth type, the range of choices progressively diminishes. In the first case (that of the *kollektive Verbindung*), the group has the maximum amount of freedom, or the minimum amount of structuring; the next grouping (by juxtaposition) requires the appropriate form of spatial or temporal connection; the grouping by *figurale Momente* requires a further supplement of structure tied at least to the *positions* of the parts; the fourth case seems to be wholly constrained as regards its dependences structure.

It therefore seems evident that the range of action of possible perspective shifters is greatest in the case of the *kollektive Verbindung* and then progressively diminishes from one case to the next. For this reason, the discussion that follows will be restricted to the first and occasionally second type of grouping.

However, on careful consideration, there is a level even more primitive than the *kollektive Verbindung*. The latter, in fact, presupposes that the elements to be connected are already given; it is not concerned with how the initial elements grouped together by the operation of *kollektive Verbindung* have been cognitively constituted. If we consider this aspect as well, we must conclude that the initial cognitive situation cannot have any objectual valence. Pre-cognitively, that is to say, there are no objects to connect. The above is a reasonable rendering of Mr Negoro's "unknown space".

This raises the question as to how one passes from this initial pre-cognitive situation to one in which the elements are placed against a ground so that they can be connected by a cognitive act of *kollektive Verbindung*.

The constitution of the figure as the effect of a grouping operation – and this brings us to the second case – is that the figure proceeds in parallel with our perspective on it. Thus, having described the operations of the first category of requirement shifters as grouping operations, I shall refer to those of this second family of requirement shifters as round-going operations.

3. Round-going

Whatever the outcome of the grouping, the figure that emerges is an incomplete figure. The phenomenological terminology talks in this regard of *Abschattung*, or that part of the object which is effectively given in a possible perception. A figure of this kind is constructed from a certain vantage point and orientation of the cognitive subject; it has its directionality, a background, etc. Modifying any of these factors changes the resulting figure. It is only possible to speak of an actual object if there is some principle which unifies the various *Abschattungen* into something that possesses the features of a whole. In this sense, every point of view is partial and must be coordinated with other views of the object. The *Abschattungen* are thus the images that derive from *round-going* (in the sense of circling around) the object.

Three series of observations are relevant in this regard. The first concerns the difference between incomplete object and absent object, the second the difference between explicit and implicit, the third the problem of objectivity. I begin with the difference between incomplete object and absent object.

3.1. Complete and absent objects

Generally speaking, it is evident from what we have seen regarding *Abschattungen* that all *represented* objects are intrinsically incomplete objects, and this is because every represented object is intrinsically or essentially connected to a specific perspective [3: 206-208].

Given that we are never able simultaneously to grasp all the perceptive and cognitive aspects of an object, even an actually presented object is always at least partly (or even largely) absent. From these premises, it follows that the concept of external or represented object is more an element experienced with the *character of completeness* than something *effectively* experienced in all its sensory and cognitive dimensions. In the strict sense, indeed, an external object is never *completely present*.

The object, moreover, is experienced as complete even though our access to it is always and exclusively through *Abschattungen*. Thus the completeness of the object assumes the character of an *implicit* closure function embedded in every *Abschattung* and which has the task of governing the *objectivity* conditions of the object. To proceed, therefore, we must consider the categories of ‘implicit’ and of ‘objectivity’.

3.2. Implicit object and explicit object

‘Implicit’ has two meanings. Under the first meaning, implicit can mean ‘unnoted’ and explicit can mean ‘noted’. Unnoted elements in the perceptual field, in spite of their being unnoted, have nevertheless a functional meaning, i.e., “have a function in the constitution of the meaning of the figure or area of focal awareness”. That things are so is proved by the fact that “this functional meaning changes when these elements are brought to specification, i.e., explicitly noted. Hence in a real sense the analysis which effects this explanation is introducing new elements rather than merely resolving a complex into its parts”. This point seems to have some relevance, because it is one of the possible reasons that explains “why Gestalten resist topological analysis” [4: 310].

The second meaning of ‘implicit’ is more difficult to explain because it involves general options of a metaphysical nature. For some, this simply signifies that these are concepts that are not yet scientifically reliable. In short, ‘implicit’ in this second sense refers to all the procedures that unify the multiple *Abschattungen* into something that has the valence of a whole. Here the two fundamental options are well known: on the one hand there is the Kantian option, according to which the unification is governed by the cognitive subject; on

the other, there is the option that I shall call, hopefully without causing offence, Aristotelian, and on the basis of which at least some unifications have an ontological foundation. For the moment I shall restrict myself to this remark, resuming to the attack later.

3.3. Objectivity

The third aspect to consider is objectivity. The objectivity of the object is the aspect which stabilizes the cognitive situation by systematically coordinating the various perspectives. There are two main possibilities here: the first is that of reciprocal coordination (or the mutual translation) among the *Abschattungen*; the second requires the coordination of the various perspectives with the so-called perspective 'from nowhere'. In the former case, the object is knowable by means of a set of *Abschattungen* formally unified by a purely functional, intrinsically unknowable bearer. This, of course, is the Kantian X. In the latter case, we know the object not also by means of its *Abschattungen* but also in the apparently paradoxical form of the object from nowhere – that is, the object as it would be if it were devoid of any perspective. This position is often erroneously taken to be a form of Platonism, whereas in fact it has nothing to do with it. It is not a Platonic position because it states neither that the object from nowhere is an ideal object nor that the *Abschattungen* are appearances. This position in effect amounts to a modern version of Aristotelianism in which the object is primarily the material bearer of its *Abschattungen* and secondarily an object formally characterized as an object from nowhere. The object as the material bearer of its *Abschattungen* is a version of the Aristotelian theory of the prime substance, while the object from nowhere performs the role that in Aristotle's philosophy is undertaken by the theory of the second substance (i.e. the theory of natural kinds). Here I certainly do not intend to reiterate Aristotle; even less do I intend to do so in one of the diverse and systematically misleading interpretations that still enjoy such high repute. For the moment, I shall merely point out the connection between my present discussion and certain aspects of Aristotle's own theories, a connection moreover which is not so far-fetched if one considers the presence of similar notions in authors like Leibniz, Bolzano, Brentano and Husserl. Whatever the case may be, let us return to the problem of objectivity.

In this regard, at minimum we may state, with René Thom, that those objects which possess at least a minimum of structural stability [5] are objective and are therefore categorizable. It is on the basis of this minimum condition of stability that the various perspectives on the object can be developed.

In this sense, we may hypothesise that the objectivity of representation is ensured by the intervention of a principle of holonomy; that is, a law whereby different aspects (representations) of the same object depend on the different positions of the subject(s) viewing it. Holonomy states that there is a connection between the change in the observer and the change in the (represented) object. If the connection is not holonymous (as in quantum mechanics), or if there is no connection, objectivity becomes difficult to define.

4. Approaching and distancing

The third group of operations, which intervene after the forms of grouping and their stabilization by some closure mechanism activated by round-going operations, are those of approaching and distancing. At a first level, relevant to this group of operations are issues like the windowing of attention and granularity. The windowing of attention allows what is thrown into relief to be separated from what remains in the background. Granularity likewise selects a pertinent level of detail. In both cases it is not difficult to hypothesise the intervention of operators which serve to shift the focus of attention (for example, from one part to another of the profiled object or from one part of the whole to the whole itself) and to adjust the granularity. These are important components of this third group of operators. I shall return to them later, but first I must consider a deeper-lying aspect of the operations of cognitive approaching and distancing. If they are not carefully analysed, in fact, the operations which shift the attention window and adjust granularity may obscure an important qualitative difference: not all attention windows and not all granularities, in fact, have the same cognitive pregnancy. Some of them are more informative, fundamental or meaningful than others.

A good point to start from in clarifying this aspect is the theory of the sculptor Hildebrand concerning the difference between vision from close up and vision from a distance [6]. Close-up vision is that in which the object is the product of an aggregate of representations and movements. Vision from a distance – which for Hildebrand is the authentically artistic sort – is that form of vision which allows unitary apprehension of the object, with a single act, and in which the volumetric characteristics of the third dimension are rendered into surface.

This is a development of the distinction introduced by Zimmermann – a pupil of Bolzano – between tactile visual representation and optical visual representation [7]. In the former case, the eye sees objects as if it were touching them (whence ‘tactile’ representation), whereas in the latter the object is presented as a whole, with the effects of which it is capable. Although it is not possible to dwell on Hildebrand’s ideas here, I shall use them at least to show that the difference between vision from a distance and vision from close up does not reduce to the fact that from close up we see details that we cannot see from a distance. The problem, that is to say, is not one of a greater or lesser amount of *local* information. The difference between the two types of vision, in fact, is not quantitative but *qualitative*. In more contemporary terms we may perhaps distinguish them as analytic vision and holistic vision.

Given this difference between vision from close up and vision from a distance, we may resume analysis of attentive selection and of granularity. Regarding granularity in particular, it is worth noting that it is typically understood to be an articulation of the problem of the levels of description. But this problem relates to and subsumes the much more fundamental one of the levels of reality [11]. This I shall discuss shortly, but first we require some further specifications.

5. Base-highlighting

I shall use the term *base-highlighting* to refer to the fourth type of requirement shifters. The reason for choosing this designation will become evident when these shifters have been seen in operation.

Unlike the previous cases, I shall exemplify this type of shifter with linguistic phenomena. The shifters in this case are eminently represented by so-called *as-phrases*, of which examples are *as a teacher* or *as a prime number*. Friedrike Moltmann has claimed that perspective shifters such as *as-phrases* “have the function of specifying the ‘basis’ for the application of the predicate” [8: 74]. When I say

John as a teacher is good

the phrase ‘as a teacher’ specifies that John is good regarding his teacher qualities. In this sense, the operator *as* “restrict the properties one may attribute to John to those he has on the basis of being a teacher” [8: 72].

Note that *as-phrases* may provide more than one basis, therefore jointly considering an entity from several points of view. Consider for instance:

John is good as a teacher and bad as a father

As-phrases may also specify the dimensions of integrity of an entity, as in

Kurdistan as a cultural unit has more integrity than Kurdistan as a political unit [8: 79].

We have in this case a clear example of a multimodal entity which possesses dimensions characterized by different degrees of integrity. This signifies that an entity like a whole is not necessarily and to the same extent a whole in all the dimensions in which it has value.

Consider furthermore that not all the basis are suitable for all the predicates. Moltmann [8: 77] provides the following examples:

- # John as a teacher was born in France
- # John as a teacher is forty years old
- # John as a teacher knocked at the door yesterday

In many cases, bases of the type *as a teacher* or *as a father* refer to situations which in the sociological literature are termed 'roles'. In this case the social agent is an entity which admits a multiplicity of different roles – with various facets – tied to different contexts of reference. The same agent displays different aspects and features according to his various life-situations: as a father, husband, professor, citizen, football fan, and so on.

This latter observation becomes of central importance to my topic as soon as one realizes the surprising correspondence between the roles and the *Abschattungen* of an entity. I obviously use the term 'correspondence' and not 'identity' because *Abschattungen* are intrinsically tied to the individual perceptive act, while roles are intersubjective structures which result from a much more complex process of formation. Nevertheless in both cases the same questions can be asked.

As regards *Abschattungen*, I asked earlier whether they require a purely formal unification of a Kantian X type, or whether they instead admit to both a material unification in an underlying bearer (or prime substance) and a parallel formal unification of typicality displaying the features of Aristotle's second substance. In entirely similar manner we may ask whether roles are formally synthesised in a bearer analogous to the Kantian X or whether they require both a material bearer and a parallel formal synthesis.

All this may seem to be a pointless complication until one notes that living systems – and therefore also social systems – have the distinctive characteristic of being able to reproduce the elements of which they are composed. It thus becomes of particular importance to identify the elements that make up social systems and which a social system, as a living system, should be able to reproduce. It is obvious that social systems – as we know them – require bearers which are simultaneously biological systems and psychological systems, but that they do not reproduce either organisms (biological systems) or minds (psychological systems). At the level of social systems that which is reproduced are roles, not organisms or minds [9].

In other words, if, as seems natural, we assume that social systems are communication-based systems, then it follows that roles – which govern communicative expectations – act as constitutive elements of social systems themselves.

The points just briefly outlined are significant for at least two reasons. First, they highlight that certain systems require other systems as their bearers, and that every system at its own level requires adequate conditions of unification. Second, they show that the entities that make up systems are stratified and have numerous internal dependence relations.

I am aware that the foregoing arguments have been outlined so briefly that they may seem somewhat impenetrable. In less intimidating terms, the following conversation between the mathematicians Gian Carlo Rota and Stanislav Ulam sets out the nucleus of the discussion so far in colloquial terms.

Ulam: "Now look at that man passing by in a car. How do you tell that it is *not just a man* you are seeing, but a *passenger*?"

"When you write down precise definitions for these words, you discover that *what you are describing is not an object, but a function, a role that is inextricably tied to some context*. Take away the context, and the meaning also disappears".

"When you perceive intelligently, as you sometimes do, *you always perceive a function, never an object in the set-theoretic or physical sense*".

"Your Cartesian idea of a device in the brain that does the registering is based upon a misleading analogy between vision and photography. Cameras always register objects, but *human perception is always the perception of functional roles*. The two processes could not be more different".

"Your friends in A.I. are now beginning to trumpet the role of contexts, but they are not practicing their lesson. They still want to build machines that see by imitating cameras, perhaps with some feedback thrown in. Such an approach is bound to fail since it starts out with a logical misunderstanding."

To this words, Rota replies: "Do you then propose that we give up mathematical logic?"

"Quite the opposite. Logic formalizes only *very few* of the processes by which we think. The time has come to enrich formal logic by adding some other fundamental notions to it. What is it that you see when you see? You see an object *as* a key, you see a man in a car *as* a passenger, you see some sheets of paper *as* a book. It is the word 'as' that must be mathematically formalized, on a par with the connectives 'and,' or,' 'implies,' and 'not' that have already been accepted into a formal logic. Until you do that, you will not get very far with your A.I. problem" [10: 57-59].

If we now turn to our requirement shifters, it is immediately evident that my general reflections and Ulam's remarks relate to entirely manifest phenomena. Consider for example a statement like

John as a teacher was born in France.

This statement is not acceptable because the basis *as a teacher* does not sustain the predication *born in France*. A statement of this kind can become acceptable in one of the two following ways. The first is to cancel the reference to a basis. In this case we obtain

John was born in France

which is obviously entirely acceptable. However, the drawback to this manner of resolving the difficulty is that the reference to the basis is lost. The most important aspect of as-phrases is in fact that they furnish us with a basis that is able (if the statement is acceptable) to support the ensuing predication. An as-phrase acts as a prism which divides predicates into classes of membership and thus makes the structure of the object somewhat more explicit. Accordingly, instead of removing the reference to a basis, we may reformulate the statement in question by looking for an alternative basis able to give meaningfulness to the statement. If

John as a teacher was born in France

is not acceptable, what we can do is to proceed, not by removing the as-phrase but by replacing the basis which does not work with one that does. That is to say, what we must do is ask ourselves what term can substitute '???' in

John as a ??? was born in France

so that a meaningful statement is obtained. In this case, one need only experiment with a few expressions to realise that no term that refers to roles can replace ???. Roles, in fact, are modifiers which typically restrict or enlarge the set of admissible predicates, whereas what is required here is a term which at minimum *does not restrict* and *does not enlarge* the set of predicates applicable to John. This means that there are predicates that attach to John independently of the roles that he may assume.

If this is true, then we have found a criterion with which to distinguish predication with as-phrases from predication without as-phrases.

The first of these forms of predication, that with as-phrases, is performed on qualifications connected with some role. The second of them concerns qualifications which are independent of roles – that is, ones *intrinsic* to the object qualified. But this is not all. By varying the examples, one notes that intrinsic qualifications – i.e. those not tied to roles – are qualifications which pertain to the physical, biological or psychological dimensions of John, but not to his social dimension. In other words, they are qualifications which do not pertain to John as a social actor but to John as a biological or psychological entity. What we now need, therefore, as we also saw in the above discussion of shifters of approach and distancing, is a theory of the layers of reality, and not solely a theory of the levels of description of reality. Unfortunately, this brief mention will have to suffice, for it is a theory too complex to be presented *en passant* [11].

6. Reduplication

While discussing the fourth type of perspective shifters, I pointed out the important role performed by as-phrases. Using a Latin expressive form, these are sometimes presented in the guise of *qua*-phases, as in

John *qua* teacher is good.

The function '*qua*' – used in expressions like '*A qua B is C*' – has a long philosophical pedigree. In fact, the word '*qua*' is the Latin translation of the Greek '*he*' in the expression '*on he on*' which in the seventeenth century gave origin to the term 'ontology'.

In the philosophical literature, *qua*-theories are usually referred to as reduplicative-theories or as theories of reduplication. Since Aristotle, *qua*-theories have been intensively used, although only in very few cases has there been explicit analysis of the theory as such.

Generally speaking, the problem of reduplication is one of the many facets of the problem of *context dependence*. There is more, however. Reduplication proves useful in clarifying the idea of the perspective from nowhere. It is obviously appropriate to call *reflexive* that form of reduplication in which the canonical structure '*A qua B is C*' assumes the form '*A qua A is C*' [12, 13]. At this point it is evident that the reduplication operator acts as a derelativization (or decontextualization) operator in which the basis highlighted by the expression '*A qua A*' is the object viewed according to the canons of the perspective from nowhere discussed above.

7. Conclusions

I may now sum up at least a part of what has been said so far. In these final notes I shall concentrate on a single concluding topic, that of the relationship between perspective dependence and objectivity.

On the one hand it seems indubitable that perspectives are intrinsically related to the point of view adopted. On the other, the objectivity of an object seems to be in some way independent of the perspectives on it. Interestingly, and despite the contentions of numerous authors, our perceptive apparatus is the instrument that more than any other objectifies the world. Of this we all have striking evidence as long as we consider the shape of a coin. When someone is asked to describe the shape of a nickel, the conventional answer is that it is round. In fact, however, only a moment's reflection is required to realize that there is only one situation in which we actually see a round coin: when it is at an appropriate distance from us and when we view it head-on, at the same height as the eyes and at the same distance from both of them. A situation which almost never occurs. In all other cases, what we should see is an ellipsoid object.

From the point of view that interests us here, the 'objectual' dimension of objects is precisely that structural invariant that unifies and underlies all its possible perspective-based variants. This is precisely the second substance of Aristotelian memory so frequently mentioned in this paper. Note that it is a valence directly embedded in perception and cognition; it is not an abstract and ideal objectuality located in some implausible celestial sphere.

It should also be noted that, unless we allow ourselves to be caught out by a superficial reading, Gibson's ecological approach acknowledges this objectuality from nowhere. In the synthesis provided by Heft, it is as if we perceive – quote – "objects and environmental layout from all sides at once" [14: 124]. A similar proposal is to be found in Merleau-Ponty [15]. Here I continue drawing on the synthesis by Heft [14: 125]. Quote: "I see the next-door house from a certain angle, but it would be seen differently from the right bank of the Seine, or again from an aeroplane: the house itself is none of these appearances; it is, as Leibniz said, *the flat projection* of these perspectives and of all possible perspectives, that is, the *perspectiveless* position from which all can be derived, *the house seen from nowhere*" (my emphasis). Two remarks are in order. First, as Heft points out, "the house seen from everywhere at once is a house seen from nowhere in particular" [14: 125]. Second, in order to obtain a view from nowhere we must 'neutralize' the perspectival components. If this is so, an objective view is an intrinsically non-perspective-based view. The problem is, how do we obtain one?

Consider the pen in front of me on my desk, next to the keyboard with which I am writing this paper. What type of object is this pen? How should I model it?

First of all, I may say that the pen is an object made in a certain way, with its own shape, colour and material. In saying these things, I use concepts that serve to describe the physical world of things. The pen must also perform functions: it has been designed to write. This reference to function, to the activity of writing, introduces a different dimension into the analysis. Writing, in fact, is not something that I can model using only concepts

describing the physical world. Writing is an activity typically performed by humans. By virtue of being constructed to fulfil the function of writing, the pen is in some way connected with this aspect of the world. But when I observe the pen, it tells me many other things. It tells me, for example, that it has been constructed by somebody, and that this somebody is my contemporary. The pen conveys information that informs me that it is not an object from the Roman age or from ancient China. On the contrary, the material of which it is made, its manufacture, its shape, the way it works, tell me that it belongs to the contemporary epoch. For all these reasons, the pen also tells me that there must somewhere be an organization that produces things like pens. If we now shift our focus to this organization, the pen must be an object designed, manufactured and distributed so that it can be sold and end up on someone's desk. In their turn, the points of view of the designer, of the production department and of the distribution department are different, and they describe my pen using different concepts. For the designer the pen is essentially an aesthetic and functional object; for the production department it is the outcome of materials processed in a certain way; for the distribution department it is something else besides. For the company producing the pen it is all these things together. For the shopkeeper who displays the pen on his shelves and seeks to sell it to customers, it is again different. To return to myself, the pen is also an object to which I am particularly attached because it reminds me of the person who gave it to me [16].

All these multiple and diverse descriptions are correct. Each of them manifests an aspect of the object before me. Yet all these descriptions are descriptions of the *same* object. Hence, one of the fundamental tasks is to find a way to *integrate those different descriptions of the same object*.

Some of these descriptions have an ontological basis; others have cognitive or epistemological bases. We must learn to distinguish among them. Ontologically, the example of the pen teaches us two important lessons: (i) reality is organized into strata (material, psychological, social); (ii) these strata are organized into layers (the physical and chemical levels of the material stratum; the intentional and emotive levels of the mental stratum; the productive, commercial and legal levels of the social stratum). For every (type of) object there must be a schema or template which coordinates and synthesises the admissible descriptions of it. And for every object the template that best characterizes it must be elaborated. This, in the case of my pen, might be the template 'artifact', which comprises the fact that the object is above all social in nature and consequently has social components ('is made by', 'for', 'costs so much'). However, these dimensions do not account for the ontological structure in its entirety. The artifact also has a material basis, and there may also be components embedded in its structure which evoke psychic components (these are the 'affordances' proposed by Gibson [17]).

For obvious reasons, an epistemology does not have the capacities of coordinating those aspects. For at least some of them we need an ontology. Only an ontology, in fact, may find a way of coordinating those aspects [18, 16, 11, 19].

In the case of the pen, various viewpoints are concretized in a single object. But this is not the only possibility. If we consider a complex like a hospital, this comprises various agents (doctors, nurses, patients, administrators, auxiliary staff) who carry out very different and complex operations and tasks. In a system of this kind, the various agents of which it is composed perform different tasks, and they have different functions and responsibilities. But they must somehow understand each other and integrate into a unitary whole. In the case of the pen, the various viewpoints converge in the object 'pen', and they do not operate independently of the object. In the case of the hospital, however, the viewpoints converge on a dimension which is apparently less concrete and which is something akin to the institutional *function* or *role* that the object 'hospital' must perform. Secondly, this function or role may be given different interpretations by the agents who make up the institution. Their various viewpoints may also generate conflicts internally to the institution, so that it can achieve its goals more efficiently if the viewpoints are adequately synthesised, or lapse into paralysis and breakdown if they are not.

This gives rise to a highly variegated pattern, a multi-dimensioned panorama with which we are only minimally acquainted. The general architecture of the situation, in fact, is in many respects still unknown.

Whatever the case may be, it is crucial to distinguish the ontological and the cognitive dimensions of objects. Let us consider the following. Suppose that four walnuts are

arranged in a rough square on a table. I may say that the walnuts form a square, or I may say that they form a cross, or I may again say that they are arranged in an 'x'. The particular constituted by the pattern assumed by the walnuts is a cognitive particular *founded* on an ontological base. The important fact is that the base permits some interpretations and not others. As a second example, let us consider a uniformly red surface (of a material object or part of a material object). If we say 'the central part of the surface' or 'the top right-hand area', these are cognitive off-cuts which are grounded on objectively given data. Between 'almost purely ontological' objects and 'almost purely cognitive' ones there will obviously be ambiguous, blurred and perhaps even undecidable cases.

Having reached this point, I may now conclude with a final observation. If we seek to divide the various components that make up a representation, we may state that any representation whatever is the representation of (1) something, (2) in a certain way, (3) from a certain perspective. We may substitute (1) with: object, process, event, organ, function, situation, institution, or with many other terms of the same type. For (2) and (3) we must draw on similar lists of modes and points of view. Leaving aside (2), we have seen at least some of items of (3). This proves that, even if it is unlikely to provide an exhaustive list of modes and viewpoints, it is nonetheless possible to identify at least some of them. In this situation, the 'perspective from nowhere' is that canonical global reading that underlines and unifies all the various perspectives on the object under description.

References

- [1] R. Poli, Automatic Generation of Programs: An Overview of Lyee Methodology. SCI2002. 2002. Retrievable from <http://www.mittleeuropafoundation.it/RP/Polipapers.htm>.
- [2] B. Smith and K. Mulligan, Pieces of a Theory. In: B. Smith (ed.), Parts and Moments. Studies in Logic and Formal Ontology. Philosophia: München, 1982, pp. 15-109.
- [3] R. Poli, Levels. Axiomathes, 9 (1998), pp. 197-211.
- [4] A. Wilden, System and Structure. New York: Tavistock Publications 1980 (2nd ed.).
- [5] R. Thom, Stabilité structurelle et morphogénèse. Paris: Ediscience, 1972.
- [6] A. Hildebrand, Das Problem der Form. Heitz: Strassburg 1905 (1893).
- [7] R. Zimmermann, Allgemeine Aesthetik als Formwissenschaft. Vienna 1865.
- [8] F. Moltmann, Parts and Wholes in Semantics. Oxford: Oxford University Press, 1997.
- [9] N. Luhmann, Soziale Systeme. Grundriss einer allgemeinen Theorie. Frankfurt. Suhrkamp. 1984.
- [10] G. C. Rota, Indiscrete Thoughts. Boston-Basel-Berlin: Birkhäuser, 1997.
- [11] R. Poli, The Basic Problem of the Theory of Levels of Reality. Axiomathes, 12 (2001), pp. 261-283.
- [12] R. Poli, Formal Aspects of Reduplication. Logic and logical philosophy (1994), pp. 87-102.
- [13] R. Poli, Qua-Theories. In: L. Albertazzi (ed.), Shapes of forms. Dordrecht: Kluwer, 1998, pp. 245-256.
- [14] H. Heft, The Ecological Approach to Navigation: A Gibsonian Perspective. In: J. Portugali (ed.), The Construction of Cognitive Maps. Dordrecht: Kluwer, 1996, pp. 105-132.
- [15] M. Merleau-Ponty, The Phenomenology of Perception. London: Routledge, 1963.
- [16] R. Poli, ALWIS. Ontology for Knowledge Engineers, PhD Thesis, Utrecht 2001.
- [17] J.J. Gibson, The Ecological Approach to Visual Perception. Boston: Houghton, 1979.
- [18] R. Poli, Ontology for Knowledge Organization. In: R. Green (ed.), Knowledge Organization and Change. Frankfurt am Main: INDEKS Verlag, 1996, pp. 313-319.
- [19] R. Poli, Ontological Methodology. Forthcoming in Journal of Human Computer Studies.

On the Philosophical Foundation of Lyee: Interaction Theories and Lyee

Bipin INDURKHYA
Tokyo University of Agriculture and Technology
Koganei, Tokyo (JAPAN)

Abstract: This paper aims to examine the philosophical foundation of Lyee. In particular, Lyee's hypotheses are compared with those of the interaction view of cognition, and a number of parallels are noted. Based on these connections, I propose some extensions and elaborations of Lyee that might make it more widely applicable to areas other than software engineering. In particular, I suggest three areas of cognitive science where Lyee methodology might be relevant. These areas are: 1) modeling the interaction of a cognitive agent with its environment, 2) modeling the social interaction of a number of agents sharing an environment, and 3) modeling creativity of cognition.

1. Introduction

Lyee is a methodology for designing and implementing software systems, which, according to its inventor, is derived from the philosophical traditions of Leibniz, Spinoza, and Wittgenstein [16, 20]. Though in the domain of software design Lyee has been shown to be remarkably successful — several ongoing projects are further exploring and evaluating the effectiveness of Lyee, comparing it with other methods and elaborating it along the way — its philosophical foundation are not clearly articulated. Various papers about Lyee contain a brief discussion of its philosophical motivation [15, 16, 17, 18, 20], but these discussions are too short to really do justice to the philosophical traditions on which Lyee is based. The most comprehensive paper about the theory underlying Lyee is perhaps [19], which contains a number of axioms and rules but there is very little in terms of motivation, background and intuitive explanations.

One may naturally ask here why bother looking under the hood of something that seems to work well. If the inventors of Lyee have hit upon a method to make large software systems intuitively, efficiently and economically, that seems a godsend. Why bother with its philosophical roots, whatever they may be — they hardly seem relevant for a practical software engineer.

However, I think there are two major reasons for exploring beneath the success stories of Lyee. One is that if Lyee seems such an effective methodology for designing large software systems, understanding its theoretical foundation is bound to reveal us something about the very nature of software itself. Needless to say, the relevance of this matter for even a practical software engineer cannot be overstated.

The other reason is that according to the inventors of Lyee, the methodology is based on the philosophical observations about the very nature of cognition. This naturally leads one to wonder if this methodology might be applicable to other domains as well. For example, can the ideas behind Lyee be used to do cognitive robotics? Can they be applied to design creativity support systems? And so on. In order to realize the full potential of Lyee methodology in domains other than software design, it is imperative that its

philosophical foundation be properly articulated and elaborated.

With this goal in mind, in this paper I undertake a tentative exploration of the philosophical foundation of Lyee and examine the feasibility of applying this approach to some problems in cognitive science. I say 'tentative' because my understanding of Lyee is only partial at the time of writing this paper. In that sense, this paper is more like a snapshot of an ongoing research project rather than the final result.

Specifically, I feel that Lyee incorporates many features of what is sometimes referred to as an interaction theory of cognition. Though there are many variants of this view in the literature, one thing that they all agree on is that a cognitive agent's conceptual structures are created in part by the agent itself and determined in part by the environment: in other words, they derive from an interaction between the two. In this paper, I will elaborate on this connection, and argue how it might provide a conduit for applying Lyee ideas to cognitive science.

This paper is organized as follows. In the next section I provide an overview of the interaction theory of cognition. In Sec. 3 the connection between Lyee and the interaction theory is explored. In Sec. 4 implications of these connections for cognitive science are discussed. Finally, Sec. 5 summarized the main conclusions of this paper and points out the future research problems.

2. Cognition as Interaction

As mentioned above, there are many versions of the interaction view of cognition, and not all of them explicitly use the term 'interaction'. I will sketch here an outline of some of its major versions, emphasizing those features that are relevant to Lyee. First of all, to provide a contrast, let me mention an opposing view of cognition, sometimes known as objectivism, according to which the world is really composed of objects, attributes, relations, and so on. These are all mind-independent, and it is the primary task of cognition to *discover* these structures. Many scientists and engineers take this viewpoint, and one may look at Deutsch [6] to find recent arguments in its support.

Though objectivism is sometimes contrasted with subjectivism, according to which there are no objective criteria and anything goes so to say, but actually it is difficult to find any scientist or scholar seriously arguing for such an extreme position. What we find is a range of viewpoints that all assert the creative aspect of cognition, but differ widely in how much role is assigned to the environment to constrain this creativity. I should emphasize that most of these theories do have some criteria incorporating objectivity built in implicitly though such criteria are given a lesser role and are not emphasized. Perhaps Feyerabend [7] may represent a view that is closer to the subjectivist end. Interaction theories generally lie somewhere in the middle.

2.1 Kant, Cassirer and Goodman: From Schemas and Symbolic Forms to Worldmaking

The origin of one strand of interaction theory may be traced back to Kant [12], who took his point of departure from the earlier traditions, where knowledge was considered to be knowledge of the objects in the world, and focused, instead, on the process of cognition, which forces objects into the structure of concepts. Kant also made a distinction between the *noumenal* world, a world of *things in themselves* that is essentially inaccessible, and a *phenomenal* world of appearances that is given to us by our senses and in which all cognition takes place. Kant referred to the concepts a cognitive agent might have as categories, which are subjective in that they highlight the agent's role in cognition, but are objective in that they do not vary from consciousness to consciousness. For Kant it was necessary to take the latter position in order to account for the objective nature of natural

sciences.

Kant introduced the notion of *schemas* that mediate between categories and the world of appearances, as the world of *things in themselves* is inaccessible. However, we should take note of Krausser's [13] arguments that there are two postulates regarding the world of *things in themselves* implied in Kant's epistemology. One is that in order to have an empirical structure in the sciences, we must be able to interact with and experience the world of things in themselves. Secondly, this world of *things in themselves* must have some structure even though this structure is not knowable. As I will argue in the next section, both these postulates are relevant for Lyee.

Kant's metaphysical revolution, in regarding only the process of cognition as accessible and knowable but not the world of *things in themselves*, achieved its full intensity in the philosophy of Ernst Cassirer [4], who developed Kant's schemas into a theory of symbolic forms and proceeded to apply it to various human activities such as language, mathematics, natural sciences, myth and religion. Cassirer saw the creation of symbols as primary function of human consciousness: it is only in making of symbols that the kaleidoscopic flux of impressions is halted and given a form that can be comprehended.

"Every authentic function of the human spirit has this decisive characteristic in common with cognition: it does not merely copy but rather embodies an original, formative power. It does not express passively the mere fact that something is present but contains an independent energy of the human spirit through which the simple presence of the phenomenon assumes a definite 'meaning,' a particular ideational content." ([4], p. 78).

This idea concerning the multiplicity of symbolic worlds in Cassirer's philosophy is further elaborated by Nelson Goodman [8], who argued that different modes of cognition create their own 'worlds', but each 'world' has to have its own objective criteria for rightness.

2.2 Piaget's Constructivism

Another strand of interaction theory that is relevant to Lyee has its origin in the work of Swiss psychologist Jean Piaget [21, 22, 23], who pioneered the view that common concepts such as object permanence, time duration, temporal succession, class inclusion, and so on, are all *constructed* by the child through increasingly complex interactions with the environment. The key points of Piaget's constructivism are perhaps best summed up by one of his translators, Robert Campbell, as follows:

"For Piaget, operative knowledge consists of *cognitive structures*. ... If knowledge consists of cognitive structures, then development comes down to what structures do. According to Piaget, operative structures never just sit there; they demand to be applied. When the knowing subject applies a cognitive structure to the environment, or *assimilates* the environment to the structure, the structure may work without modification; it may fulfill the knower's goals as expected. But in some cases assimilation will be unsuccessful. In those cases, applying the scheme will not lead to attaining the goal as expected. ... When unsuccessful assimilation has taken place, the child needs to modify the scheme, in order to *accommodate* it to the environment. Accommodating the scheme to the environment may mean putting restrictions on it ... or differentiating it into one or more subschemes. On occasion, entirely new schemes will need to be constructed if successful accommodation is to take place.

Piaget believed that schemes tend to get applied, so assimilation and accommodation happen naturally. He regarded development as driven by processes that tend toward a balance, or equilibrium, between assimilation and accommodation." ([3], pp. 3–4, all emphasis original.)

There are three main features of Piaget's theory of cognition that are relevant to Lyee. One is that the cognitive structures are not just sitting there in the environment ready to be acquired, but are actively constructed by the agent. The second is the operational approach to cognition so that to know an object is to act on it. Finally, there is the articulation of the mechanisms of assimilation and accommodation that act in consort to make cognition possible.

I should add here that recent research has further emphasized the creative role played by the agent in perception and cognition (see, for instance, [9]). Also, for a more thorough discussion of the interaction view, and its application to modeling the creativity in cognition, particularly that of metaphor, the reader is referred to my earlier work [11].

3. Interaction Theories and Lyee

I will now attempt to draw a connection between the Lyee hypotheses and the interaction view. I should, however, once again emphasize that the remarks in this section are based on my current understanding of Lyee, which is only partial at best. Consequently, I offer my comments somewhat tentatively.

The first obvious connection is that Lyee explicitly recognizes the role of cognitive agent in creating objects and structures in the *natural space* that are connected to the ideas in the *consciousness space* via the agent's intentions. Lyee makes the overt assumption that the consciousness space contains the wholes that can only be comprehended by objectifying parts of them in the natural space. Moreover, it is also emphasized in Lyee that the whole asserts itself by constraining how any part of it can be objectified and what attributes it can take. This feature reflects, in my opinion, objectivity of the Lyee framework. (The reader may recall the discussion of Krausser above. See also [11], Chap. 5, Sec. 4.) I think that Lyee's consciousness space can be likened to Kant's noumenal world (the world of *things in themselves*) and the natural space to the phenomenal world (the world of appearances). With this identification, Lyee's requirements may well correspond to Kant's categories, and its Scenario Function (SF) to schemas.

The things (or units) in the consciousness space are considered to be <substance, attribute-list>, where the substance is an element of the attribute-list. The reason for this seems to be that the substance as such is considered to belong to the world of *things in themselves*, and therefore not accessible for cognition. The attributes, on the other hand, can be represented by objects in the natural space, and are therefore available for cognition.

The objects in the natural space are also considered to be <substance, attribute-list>, where the elements of the attribute-list are other objects in the natural space. Thus, Lyee does not make any distinction of sorts between the objects and attributes, which is perhaps to be expected given its noun-based interface.

In connecting the consciousness space with the natural space, Lyee recognizes that the two structures should cohere together, which is referred to as the synchronicity condition. Intuitively, it seems to say that the structure of the objects instantiated in the natural space should follow the structure of the ideas or intentions in the consciousness space, otherwise the correspondence between the two spaces is not correct. Interaction theories also contain some criterion or other for making sure that cognitive structures are coherent with the environment. (See, for example, [11], Chap. 5, Sec. 5.) This is again another facet of Lyee where objectivity is implied.

If we overlook the details and consider the three pallets of Lyee somewhat abstractly from a cognitive point of view, they seem to reflect the following architecture: the W02 pallet corresponds to the sensory organs, the W04 pallet corresponds to the effector or motor organs, and the W03 pallet corresponds to the cognitive structures. With this identification, the sequence of pallet activation by the Scenario Function in the Process

Route Diagram becomes as follows: Output \rightarrow Input \rightarrow Structural Reorganization \rightarrow Output \rightarrow ... This seems to reflect a Piagetian schema in which a cognitive agent acts on the environment, observes the results of its actions (whether expectations were met or not), reorganizes its conceptual structures if necessary, acts again on the environment, and so on.

Though at a superficial glance, the noun-based architecture of Lyee seems to be in contrast with the operational approach that is the hallmark of Piagetian constructivism, a deeper investigation reveals that the Lyee architecture is actually operational in spirit. For example, in Lyee, when N objects have been created, the number of possible requirements that can be obtained from them is N^N , which is the number of possible n -tuples that can be made from N objects. Thus, requirements in Lyee may correspond to n -tuples. Now an $(n-1)$ -place operation on N -objects can also be written as a set of n -tuples. In this way, I believe that it may be possible to express Lyee methodology algebraically. For example, some of my earlier work on formalizing a certain kind analogy using algebraic notions [10] may be relevant here.

4. Lyee and Cognitive Science Research

Having sketched out some tentative connections between Lyee and the interaction theories, I will make some remarks here on the relevance of Lyee for cognitive science research. Now for software engineering, Lyee method claims to offer a crucial advantage in doing away with the steps of writing down requirements and specifications. Similarly, for cognitive science, it should be able to eliminate the step of designing representations. In fact, one critique of most traditional approaches to cognitive science and artificial intelligence is that they suffer from *encodingism* in that they assume that certain internal state of the agent corresponds to, represents, or encodes some external object or the state of the environment. (See, for example, [2]) Lyee approach, by anchoring the intentions of the agent directly into the actions taken in the environment, without having to first work out what represents what, may offer a solution to this problem.

Of course, when we talk about the intentions of a cognitive agent here, we mean an artificial agent, a robot for example. Lyee is designed to work with a human user, whose intentions are realized in a computer program. For applying it to an artificial cognitive agent, we need to design a meta-level system that allows the agent to act with its environment using Lyee. This may require some extensions and elaborations to the hypotheses underlying Lyee. In particular, the dialectic and equilibrium of assimilation and accommodation needs to be incorporated in Lyee. In this regard, the dialectic of Lakatos [14] may be closer to Lyee's current application domain of software engineering.

Another arena of cognitive science where Lyee might be applied is that of social cognition, which concerns multiple agents inhabiting the same environment and interacting with each other. There are two aspects of this research. The first one, which may be relevant for software engineering also, is to model collective intention of a group. Philosophically, the roots of this idea may be traced back to Jung's collective consciousness. For software engineering, it means to model and objectify the collective intention of a group of engineers engaged in designing a project together. I am not sure how far the current implementation of Lyee is able to support this, and how much innovation is necessary.

The second aspect of modeling social cognition concerns the ability to explicitly incorporate other intentional agents in the model realized in Lyee's natural space. Indeed, though Piaget articulated his framework essentially in the context of an agent interacting alone with the environment, others have elaborated it to take into account emergent complex social behavior. (See, for example, [1].) In recent years, there has been an increasing interest in the AI and robotics research community to apply these ideas to model social interaction among artificial cognitive agents [5, 25, 26]. However, in order to be

applicable in this area, Lyee methodology has to be elaborated in significant ways. In particular, it has to be able to apply to itself recursively so that an agent can put itself in another agent's shoes, so to say, model its intentions and consider things from its perspective. Needless to say, if such elaborations are successful, they are likely to have a significant impact on social robotics.

Finally, I would like to mention one more area of cognitive science that is closest to my own past research, namely modeling human creativity and designing creativity-support systems. In my past research [11], I have argued that assimilation, sometimes also known as projection, plays a key role in creative problem solving and metaphor. Assimilation is also necessary to incorporate feelings and imagination in the model, which gives it a kind of 2nd order intentionality. This argument is perhaps best articulated in the following quote by Paul Ricoeur:

"Feelings, first, accompany and complete imagination in its function of *schematization* of the new predicative congruence. This schematization [...] is a kind of insight into the mixture of 'like' and 'unlike' proper to similarity. Now we may say that this instantaneous grasping of the new congruence is 'felt' as well as 'seen.' By saying that it is felt, we underscore the fact that we are included in the process as knowing subjects. If the process can be called [...] predicative *assimilation*, it is true that *we* are assimilated, that is made similar to what is seen as similar. This self-assimilation is a part of the commitment proper to the 'illocutionary' force of the metaphor as speech act. We feel *like* what we see *like*."

If we are somewhat reluctant to acknowledge this contribution of feeling to the illocutionary act of metaphorical statements, it is because we keep applying to feeling our usual interpretation of emotion as both inner and bodily states. We then miss the specific structure of feeling. As Stephen Strasser shows in *Das Gemut* [The Heart], a feeling is a second-order intentional structure. It is a process of interiorization succeeding a movement of intentional transcendence directed toward some objective state of affairs. To *feel*, in the emotional sense of the word, is to make *ours* what has been put at a distance by thought in its objectifying phase. Feelings, therefore, have a very complex kind of intentionality. ... [Their] function is to abolish the distance between knower and known without canceling the cognitive structure of thought and the intentional distance which it implies." ([24], pp.243–244, all emphasis original.)

I feel that this may be the most fruitful direction in which Lyee can lend itself to cognitive science research.

5. Conclusions and Further Research Issues

In this paper I have attempted to examine the philosophical foundation of Lyee. I have argued that the hypotheses underlying Lyee seem to have much in common with the interaction view of cognition. On the basis of these connections, I have outlined some possible avenues along which Lyee can be elaborated and applied to cognitive science research. In particular, I have identified three areas: 1) modeling the dialectic of cognition for an artificial agent; 2) modeling social cognition, where a number of interacting agent share an environment; and 3) incorporating imagination and feeling in order to model human creativity.

However, the connections sketched here are but the tip of the iceberg. As this project proceeds, my understanding of Lyee deepens, and Lyee method is elaborated further, we may find other arenas where Lyee may be applied as successfully as it has been to software design.

References

- [1] Arbib, M.A. 1985. *In Search of the Person*. The Univ. of Massachusetts Press: Amherst (Mass., USA).
- [2] Bickhard, M.H. and Terveen, L. 1995. *Foundational Issues in Artificial Intelligence and Cognitive Science*. Elsevier: Amsterdam.
- [3] Campbell R.L. 2001. Reflecting Abstraction in Context. In J. Piaget. *Studies in Reflecting Abstraction*, 1–27. R.L. Campbell (ed. & trans.) Psychology Press: E. Sussex (UK).
- [4] Cassirer, E. 1955. *The Philosophy of Symbolic Forms. Vol. I: Language*. R. Manheim (trans.) Yale Univ. Press: New Haven (Conn., USA).
- [5] Dautenhan, K. 1998. Embodiment and Interaction in Socially Intelligent Life-Like Agents. In C.L. Nehaniv (ed.) *Computation for Metaphors, Analogy, and Agents (Lecture Notes in Artificial Intelligence 1562)*, 102–142. Springer-Verlag: Berlin.
- [6] Deutsch, D. 1997. *The Fabric of Reality*. Penguin: New York.
- [7] Feyerabend, P. 1975. *Against Method*. NLB: London.
- [8] Goodman, N. 1978. *Ways of Worldmaking*. Hackett Publishing Co.: Indianapolis (Ind., USA).
- [9] Hoffman, D.D. 1998. *Visual Intelligence*. W.W. Norton & Co.: New York.
- [10] Indurkha, B. 1991. On the Role of Interpretive Analogy in Learning. *New Generation Computing* 8, 385–402.
- [11] Indurkha, B. 1992. *Metaphor and Cognition*. Kluwer: Dordrecht.
- [12] Kant, I., 1787. *Critique of Pure Reason*. N. Kemp Smith (trans.) St. Martin's Press (1965): New York.
- [13] Krausser, P. 1974. Kant's Theory of the Structure of Empirical Scientific Inquiry and Two Implied Postulates Regarding Things in Themselves. In L.W. Beck (ed.) *Kant's Theory of Knowledge*, 159–165. D. Reidel: Dordrecht (The Netherlands).
- [14] Lakatos, I. 1976. *Proofs and Refutations: The Logic of Mathematical Discovery*. Cambridge Univ. Press: Cambridge (UK).
- [15] Negoro, F. 2000. Principle of Lyee Software. *Proceedings of the 2000 International Conference on Information Society in the 21st Century (IS2000)*. Aizu, Japan.
- [16] Negoro, F. 2001. The Predicate Structure to Represent the Intention for Software. *Proceedings of the ACIS 2nd International Conference on Software Engineering, Artificial Intelligence, Networking & Parallel/Distributed Computing (SNPD'01)*. Nagoya, Japan.
- [17] Negoro, F. 2001. A Proposal for Requirements Engineering. *Proceedings of ADBIS 2001*, Vilnius, Lithuania.
- [18] Negoro, F. 2001. Methodology to Determine Software in a Deterministic Manner. *Proceeding of ICII 2001*, Beijing, China.
- [19] Negoro, F. 2002. Lyee's Hypothetical World. *This Volume*.
- [20] Negoro, F. and Hamid, I. 2001. A Proposal for Intention Engineering. *Proceedings of International Conference on Advances in Infrastructure for Electronic Business, Science and Education on the Internet (SSGRR2001)*. L'Aquila, Italy.
- [21] Piaget, J. 1953. *The Origin of Intelligence in the Child*. M. Cook (trans.) Penguin (1977): New York.
- [22] Piaget, J., 1970. *Genetic Epistemology*. E. Duckworth (trans.) Columbia Univ. Press: New York.
- [23] Piaget, J. 2001. *Studies in Reflecting Abstraction*. R.L. Campbell (ed. & trans.) Psychology Press: E. Sussex (UK).
- [24] Ricoeur P. 1978. The Metaphorical Process as Cognition, Imagination and Feeling. *Critical Inquiry* 5, 143–159. Reprinted in M. Johnson (ed.) *Philosophical Perspectives on Metaphor*, 228–247. Univ. of Minnesota Press (1981): Minneapolis (Minn., USA).
- [25] Stojanov, G. 2001. Petitagé: A Case Study in Developmental Robotics. *Proceedings of Epigenetic Robotics 2001*, Lund, Sweden.
- [26] Tani, J. 1996. Model-Based Learning for Mobile Robot Navigation from the Dynamical Systems Perspective. *IEEE Trans. On Systems, Man and Cybernetics, Part B*, Vol. 26, No. 3, 421–436.

This page intentionally left blank

Chapter 2

Software Architecture and Software Intentional Models

This page intentionally left blank

Architecture underlying the Lyee method: Analysis and Evaluation

A. Ramdane-Cherif, L. Hazem and N. Levy
PRISM, Université de Versailles St.-Quentin,
45, Avenue des Etats-Unis,
78035 Versailles Cedex, France
[*rca@prism.uvsq.fr*](mailto:rca@prism.uvsq.fr)

Abstract : Only recently the use of software architecture, has grown up considerably for the construction of reliable evolutionary systems. The structure of these systems is dynamic and continuously changing. Consequently, architectures must have the ability to react to events and perform architectural changes autonomously. In this paper our first objective is to find the style of the architecture underlying the Lyee method. Then we will analyze this architecture and show if the Lyee method is able to achieve dynamic changes. This paper describes work in progress which will lead in the future to specify precisely the structure and the behavior of the Lyee method architecture and to prove that this architecture satisfies the desired structural and behavioral properties.

Keywords: software architecture, Lyee method, intention, requirements.

1. Introduction

Lyee [1] is a methodology that source programs enables to be produced directly from an intention. This is the basic concept of the method. Lyee defines a software as a set of words and each word is further defined by a fixed number of attributes. Once the set of words is defined, it is possible to generate the corresponding program. So, software development becomes an automatic routine which does not require any creative work. The requirements are determined by users. Once user's intention is expressed, an engineer can define an almost unique set of words. Given the definitions, the program generation is simply a mechanical transformation. The methodology has been provided with an environment that is being largely used in the industry.

The Lyee method as every system has an architecture. However, the architecture is not explicitly described. An architecture may be a good one or a bad one for a given system. It will allow or prevent the system from meeting its behavioral, performance, and life-cycle requirements. Thus architecture analysis and evaluation is an important issue. Software architecture provides a means for analysis. It is a common abstraction of the system that stakeholders can use as a basis for mutual understanding and forming consensus. Fortunately, it is possible to make quality predictions about a system. These will be based solely on an evaluation of its architecture.

Good architectural design has always been a major factor in determining the success of a software system. A critical challenge faced by the developer of a software architecture is to understand whether the components of an architecture correctly integrate and satisfy all the architectural constraints and required qualities. However, it is important to provide a

method that operates at the architectural level that will provide substantial help in detecting and preventing errors early in the development [2].

Our objective, in this paper, is to find the style of the architecture underlying the Lyee method. Then we will analyze it and show if the Lyee method is able to achieve dynamic changes. This paper describes work in progress which will lead in the future to specify precisely the structure and the behavior of the Lyee architecture and to prove rigorously that this architecture satisfies the desired structural and behavioral properties for complex distributed systems.

In this paper, in the second section, we explain what we mean by the term software architecture and we focus particularly on dynamic changes of the Lyee method. In the next section, we give the global architecture underlying the Lyee method and we associate it to a known architectural style. Then, we analyze this style with respect to some quality attributes. After, we show that the Lyee method satisfies the properties of dynamic changes. Finally, we conclude with future directions for this work.

2. Software architecture

2.1 Brief Background

The “architecture” term conveys several meanings, sometimes contradictory. In our research we consider that architecture deals with the structure of the components of a system, their interrelationships and guidelines governing their design and evolution over time [3][4]. The description of an architecture is composed of identifiable components of various distinct types. Components interact in identifiable distinct ways. Connectors mediate interactions among components, they establish the rules that govern components interactions and specify any auxiliary mechanisms required. A configuration defines topologies of components and connectors. Both components and connectors have interfaces. A component interface is defined by a set of ports, which determines the component’s points of interaction with its environment. A connector interface is defined as a set of roles, which identifies the participants of an interaction. A binding defines the correspondence between elements of an internal configuration and the external interface of a component or a connector. Bindings identify equivalence between two interface points. Moreover, a connector always associates a role with a port, while a binding associates a port with another port, or a role with another role (figure-1-).

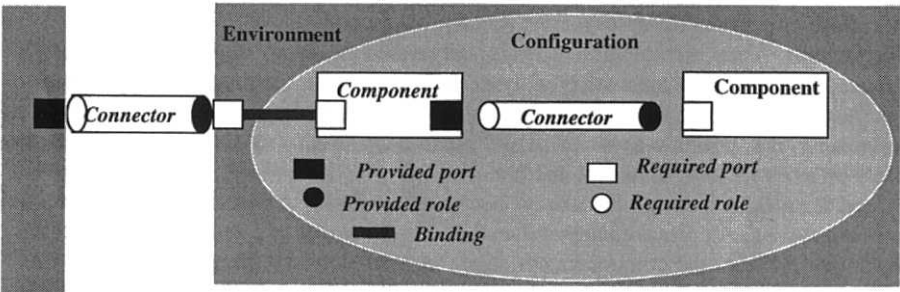


Figure-1- Software Architecture

The architectural model of a system provides a high level description that enables compositional design and analysis of components-based systems. The architecture then becomes the basis of systematic development and evolution of software systems. Furthermore, the development of complex software systems is demanding well-established approaches that guarantee the robustness and others qualities of products. This need is becoming more and more relevant as the requirements of customers and the potential of computer telecommunication networks grow. A software architecture-driven development process based on architectural styles consists of a requirement analysis phase, a software architecture phase, a design phase and maintenance and modifications phase. During the software architecture phase which we present in figure-2-, one models the system architecture. To do so, a modeling technique must be chosen, then a software architectural style must be selected and instantiated for the concrete problem to be solved [5][6]. The architecture obtained is then refined either by adding some details or by decomposing components or connectors (recursively going through modeling, choice of a style, instantiation and refinement). This process should result in an architecture that is defined abstract and reusable. The refinement produces a concrete architecture meeting the environments, the functional and non-functional requirements and all the constraints on dynamics aspect besides the static ones.

The software architecture is an abstract view of the system seen as a topology of components and connectors. It is represented by a set of architectural views. To be specific, we need more precise information on:

- Component functionality,
- Specifications,
- Properties,
- Interfaces,
- Data links,
- Control links,
- Coordination model.

Systems can have more than one architectural view. The architecture is not a single view. For example, we need:

- A conceptual (functional) view: for understanding
- A software (design) views: for work assignment, design for change
- A process (dynamic) view: for parallelization, performance modeling
- A physical (system/hardware) view: for installation, delivery, telecommunications

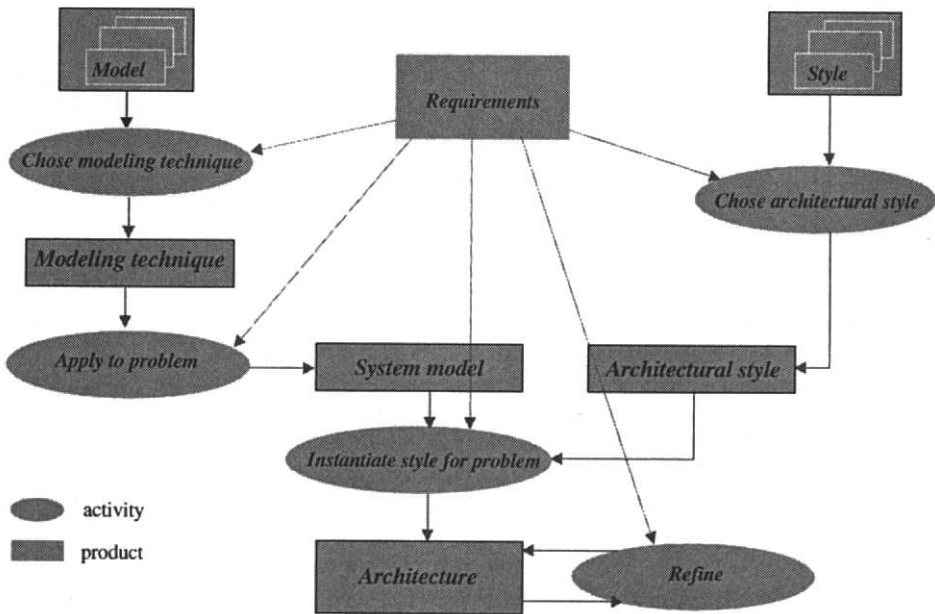


Figure-2- Software architecture phase

2.2 Dynamic architecture

In earlier works on description and analysis of architectural structures the focus has been on static architectures. Recently, the need of the specification of the dynamic aspects besides the static ones has increased [7][8]. Several authors have developed some approaches on dynamism in architectures which fulfill the important separation of the dynamic reconfiguration behavior from the non-reconfiguration. These approaches increase the reusability of certain systems components and simplify understanding [9][10].

The dynamic modifications of an architecture include the following types of changes that can occur at architectural level [11][12]:

- addition/deletion of new components,
- replacement of existing components by new updates
- reconfiguration and/or extension of the application architecture in terms of addition or deletion of connectors and components bind to those connectors.

Mechanisms like component addition (on local or on a distant site), component deletion and binding modification enable simple reconfiguration. Complex reconfigurations are implemented by composing these mechanisms but also require some others. For instance, migration consists in deleting the bindings to and from some components, saving the current state of the reconfiguration target and recreating it on a remote context before adding new bindings.

Dynamic adaptation are run-time changes depending on the execution context. These changes can be roughly classified into two categories: i) programmed reconfiguration which are part of the system design and are scheduled to happen when certain conditions are satisfied during application execution and ii) evolutionary reconfiguration which refers to structure modification specifications. Despite the differences in the nature of these

reconfiguration classes, the primitive operations that should be provided by the reconfiguration service are the same in both cases: creation and removal of components, creation and removal of links, and states transfer among components. In addition, the requirements are placed on the use of these primitives to perform a reconfiguration; to preserve all architectural constraints and to provide additional safety guarantees.

Building an adaptive architectural application is difficult because of components dependencies and of the fact components must also be able to adapt their behavior to different execution contexts. However, these architectures must have the ability to react to events and perform architectural changes autonomously. The reactivity will make application more autonomous and adaptable to changes in both the application environment and to internal events which require dynamic reconfiguration.

2.3 Adaptation process

The architecture adaptation process may entail the use of large number of data manipulation and analysis techniques and new techniques must developed on an ongoing basis. A challenge for the effective use of dynamism is coordinating the use of these techniques, which may be highly specialized, conditional and contingent and insuring that properties and qualities of the architecture are maintained and the adaptation process is properly controlled and coordinated.

The major problems that arise in considering the architecture modifiability or maintainability is:

1. evaluating the change to determine what properties are affected, what mismatches and inconsistencies may result
2. management of change to ensure protection of global properties as new components and connections are dynamically added to or deleted from the system.

3. Lyee Architecture

Since every system has an architecture, then finding the architecture underlying Lyee method is an important issue.

Currently we have only limited ways in which to state the global architecture of Lyee method and to analyze this architecture for some quality attributes. The Lyee architecture will constitute a relatively small, intellectually graspable model for how a system is structured and how its components work together. This model will be transferable across systems; in particular, it can be applied to other systems and can promote large-scale reuse.

The architecture underlying the Lyee method is presented in Figure-3-

- *Input* : requirement (words) arrives to the port 11 of the component 1;
- *Component 1* : forms a set of words;
- *Connector 1* : transmits the set of word;
- *Component 2* : proceeds to make the set of words into unit for each screen and printout;
- *Connector 2* : transmits a set of units;
- *Component 3* : establishes a SF using a unit;
- *Connector 3* : transmits plural SF;
- *Component 4* : uses four rules to link SF each other (continuous, recursive, duplicate and multiplex link);
- *Connector 4* : transmits the linked SFs;
- *Component 5* : transforms Linked SF to a unit of software, which can be defined in a diagram :Process Route Diagram (PRD);

- *Connector 5* : transfer the results of the PRD mechanisms;
- *Component 6* : automatic generation of programs and adjustment;
- *Output* : the program corresponding to the initial intention.

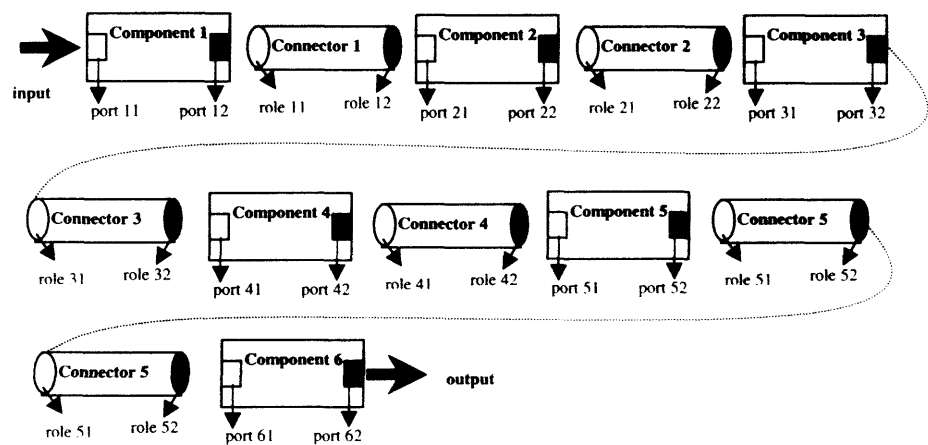


Figure-3- The architecture of the Lyee method

3.1 Analysis of the architecture

From the previous definition, we consider that the Lyee architecture looks as the famous Pipes and Filters architectural style (Pipeline). In this style each component has a set of inputs. A component reads streams of data on its inputs and produces streams of data on its outputs. This is usually accomplished by applying a local transformation to the input streams and computing incrementally, so that output begins before input is consumed. Hence components are termed filters. The connectors of this style serve as conduits for the streams, transmitting outputs of one filter to inputs of another. Hence the connectors are termed pipes. Among the important invariants of this style is the condition that filters must be independent entities: in particular, they should not share state with other filters. Another important invariant is that filters do not know the identity of their upstream and downstream filters. Furthermore, correctness of the output of this style network should not depend on the order in which the filters perform their incremental processing. This style have a number of nice properties. First, it allows the designer to understand the overall input/output behavior of a system as a simple composition of the behaviors of the individual filters. Second, it supports reuse: any two filters can be hooked together, provided they agree on the data that are being transmitted between them. Third, systems are easy to maintain and enhance: new filters can be added to existing systems and old filters can be replaced by improved ones. Fourth, it permits certain kinds of specialized analysis, such as throughput and deadlock analysis. Finally, it naturally supports concurrent execution. Each filter can be implemented as a separate task and potentially executed in parallel with other filters.

Others qualities of the Lyee architecture compared to the conventional Pipes and Filters architectures are : productivity, maintainability, optimization in the data flow: the amount of information necessary is small and simple. reduction of the complexity of human work

of dealing with requirements, correctness of SF is guaranteed, facility of defining of the process to capture intention.

3.2 Dynamic changes of the Lyee software model

The model of capturing intention is the metaphysical model which is divided into two types as shown in figure-4-. The model that establishes an axiomatic system defines intention with axiomatic system. The model that realize intention can be expressed in any programming language in this case is called Scenario Function (SF). SF can realize a software model.

If dynamic changes occur in the Lyee method, its architecture will have the ability to react to events (new intention) and to perform changes autonomously. The process will express the new intention using any program language, then we obtain the new SF that will produce the new software model Figure-4-.

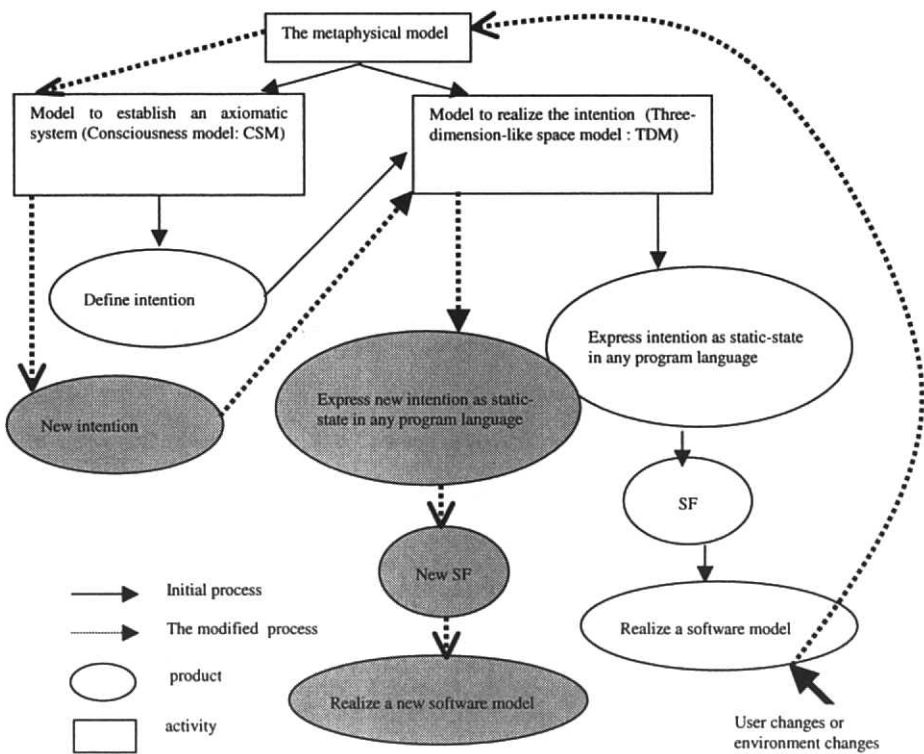


Figure-4- The metaphysical model and dynamic changes

4. Conclusion

At the beginning of this paper, we introduced the Lyee Method. Then, the notion of architecture has been explained and we introduced our research in the field of the dynamic architectures. Next, with the aim of understanding the Lyee method, we presented its global architecture. The Pipe&Filter Style was proposed as the most suitable to the Lyee method. But, we have noticed that some of the advantages of the Lyee architecture are not specific to the standard Pipe&Filter Style.

Since the Lyee method is based on intention to produce programs, the characteristics of the dynamic changes are completely automated. This quality is obvious in the Lyee method due to its particularity of iterative process to meet the user intention and also in the mechanism to produce the software model from SF.

Since every system has an architecture, then in this paper we are interested to find the architecture underlying Lyee method. This architecture will be an important issue. It permit us to study the quality attributes of the Lyee method compared to the conventional methods. This paper describes work in progress which will lead in the future to specify precisely the structure and the behavior of the Lyee architecture and to prove rigorously that this architecture satisfies the desired structural and behavioral properties. We hope in the future show that the architecture of complex distributed applications produced by applying the Lyee method can satisfy dynamic architecture concepts. These concepts were mentioned in the paragraphs 2.2 and 2.3 of this paper.

References

- [1] Negoro, Fumio : Principle of Lyee Software, Proceedings of 2000 International Conference on Information Society in the 21 Century, pp. 441-446, IS2000 Aizu, Japan, Nov. 2000.
- [2] F. Losavio, A. Matteo, Jr. O. Ordaz, N. Levy and R. Marcano-Kamenoff, Quality Characteristics to Select an Architecture for Real-Time Internet Applications. In Software and Internet, Quality Week Europe, QWE 2000, Brussels, Belgium, Nov 2000.
- [3] M. Shaw, D. Garlan, Software Architecture, Perspectives on Emerging Discipline, Prentice-Hall, Inc. , Upper Saddle River, New Jersey, 1996.
- [4] D. E. Perry, A. L. Wolf, Foundations for the Study of Software Architecture, Software Engineering Notes, 17(4):40, Oct. 1992.
- [5] N. Levy and F. Losavio. Analyzing and comparing architectural styles. In Proceedings of the XIX International Conference of the Chilean Computer Society SCCC'99, pages 87-95, 1999.
- [6] N. Levy, F. Losavio, and A. Matteo. Comparing architectural styles: Broker specializes mediator. In J. Magee and D. Perry editors, editors, Proceedings of the Third International Software Architecture Workshop (ISAW 3), pages 93-96, Orlando, Florida, USA, Nov 1998. SIGSOFT, ACM Press
- [7] P. Oreizy, N. Medvidovic, R. N. Taylor, Architecture-Based Runtime Software Evolution, Proc. 20 th Int'l Conf. On Soft. Eng. (ICSE'98), pp. 177-186, Kyoto, Japan, Apr. 1998.
- [8] J. M. Purtilo, The Polyolith Software Bus. ACM TOPLAS, 16(1):151-174, 1994.
- [9] M. Barbacci, C. Weinstock, D. Doubleday, M. Gardner, R. Lichota, Durra: A Structure Description Language for Developing Distributed Applications, IEEE Software Engineering Journal, pages 38-94, mar. 1993.
- [10] T. Bloom, M. Day, Reconfiguration and Module Replacement in Argus: Theory and Practice, IEEE Software Engineering Journal, pages 102-108, mar. 1993.
- [11] A. Ramdane-Cherif, N. Levy and F. Losavio, Adaptability of the Software Architecture Guided by the Improvement of Some Desired Software Quality Attributes. In In SERP'02: International Conference on Software Engineering Research and Practice. Monte Carlo Resort, Las Vegas, Nevada, USA, June 24-27, 2002.
- [12] A. Ramdane-Cherif, N. Levy and F. Losavio, Dynamic reconfigurable Software architecture: analysis and evaluation, In WICSA'02, The Third Working IEEE/IFIP Conference on Software Architecture. Montreal, Canada. August 25-31, 2002.

A Word-unit-based Program : Its Mathematical Structure model and actual application

Osamu ARAI,
CATENA Co, Tama-city, Tokyo, Japan,
and Hamido FUJITA,
Iwate Prefectural University, Iwate, Japan

Abstract. In the previous paper [1], the mathematical structure model of the program has been built, and the mathematical correctness has been verified. By introducing a concept of *Word*, we are thus allowed to focus on which value should be put in a region of the Word under some defined conditions in the applications. These are the characteristics of the Lyee [4]. Here, the main part of the model is mentioned again including some improvement such as mathematical model of tense control function, and an interface with people and external device when manufacturing an actual program are discussed.

1. Introduction

We think generally that a program can be defined as follows,

- ① declaration of the region, memory allocation and its related address of constants and variables
- ② calculation order which represent control sequence
- ③ computation run that program determine a sort of value assigned for memory allocation

When application is developed, a user is involved only in ③, and a system engineer and a programmer take charge of the rest of the other parts. Moreover, we input data from the screen and the keyboard by a program which is controlling the computer itself, and read and write data in the file. Specially, fixed structure exists in the part of ②, and we can generate a program automatically only by deciding (i.e., defining) ③. This structure is said to be as "the structure of the program" on Lyee's method. It is the purpose of this paper to represent this structure mathematically, to show its correctness and to show its non-sequential property.

We usually use logical relations like Hoar's logic and λ calculus to represent the program structure [2][3]. This style program structure representation help us to understand the behavioral structure of those program, so that we can understand and analyze those program specially in case of declarative style program. But the analysis done through such approach wakes its difficult to verify the semantics and/or modify it due to the complexity of such type of logical representation. In this paper we use another approach came from directed graph technologies.

2. Directed graph and its adjoin matrix

We introduce the concept of word which have region, identifier and "state". Words represent a piece of requirement in the program and region is memory allocation where the

value of this word is memorized and identifier is name of words. The state has tri-states, i.e. truth, falsehood, and the undecided.

When f_i is any program composed of x_n where x_n is the words, n is any integer and number of the words, word x_i is defined as follows.

$$\begin{aligned}
 x_i &:= f_i(x_1, x_2, \dots, x_n) \\
 &\text{where } n \text{ is number of word in system} \\
 &i = 1, 2, \dots, n \\
 &:= \text{means that program } f_i \text{ is executed} \\
 &\text{using } x_1, x_2, \dots, x_n, \text{ then value} \\
 &\text{and state of } x_i \text{ is assigned to a region} \\
 &\text{named by identifier.}
 \end{aligned}
 \tag{2-1}$$

In each equation (assignment function), words that are used on the right side are called *start-point words*. Words that are used on the left side of equation (assignment function) are called *end-point words*. We call start-point and end-point respectively forward.

The relation between start-point and end-point can be written by using directed graph in which each word is written by node and relation is written by arrow. An arrow is used to show the relation from start-point to end-point. And it also defines order that have relation $a = b$ (b is made from a).

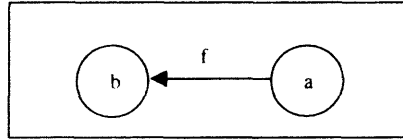


Fig. 2-1. Directed graph

Words make partial ordered set which have

relation $a = b$. Directed graph can be expressed with *adjoin matrix*. We call $n \times n$ matrix adjoin matrix F with n end-point in row, and n start-point in column. Element f_{ij} ($i = 1, 2, \dots, n; j = 1, 2, \dots, n$) of F is 1 when i -th equation of Equation (2-1) uses the x_j as start-point word, if not 0.

The relation between start-point and end-point can be written by using directed graph in which each word is written by node and relation is written by arrow.

(Example 1) x_1, x_2 as input word (terminal end-point word that have not f_i), x_3, x_4, x_5, x_6, x_7 as state word (without input word), we defined the equation (2-2) as follows

$$\begin{aligned}
 x_1 &= a_0 \\
 x_2 &= b_0 \\
 x_3 &= f_1(x_4, x_5) \\
 x_4 &= f_4(x_2, x_5) \\
 x_5 &= f_5(x_1) \\
 x_6 &= f_6(x_3, x_4) \\
 x_7 &= f_7(x_4, x_3)
 \end{aligned}
 \tag{2-2}$$

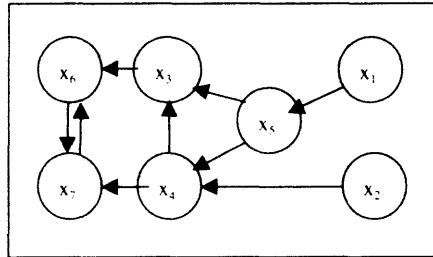


Fig.2-2 Directed graph of example

Adjoin matrix is shown as follows.

$$F = \begin{matrix} & x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & x_7 \\ \begin{matrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{matrix} & \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 \end{bmatrix} \end{matrix} \quad (2-3)$$

3. The state model for a end-point word

We record the state of the n words in the vector X (matrix of one column). A state is recorded in X in row of the order of the word in the column direction of the adjoint matrix F . The state which a word can be taken from is set at three kinds as follows.

- 1: Value is decided, truth.
- 1: Value is decided, falsehood.
- null: Value is undecided.(It isn't fixed.)

The initial value of the input word is

- 1 : Value is decided, truth.
(The state being inputted properly.)
- 1 : Value is decided, falsehood .
(The state which isn't being inputted properly.)

So, state doesn't change even if an equation (2-1) is carried out.

The initial value of the state word is,

- null: Value hasn't been decided yet.
(The state which isn't being moved or calculated yet.)

As for state word, to act a state vector on the adjoint matrix is to execute the equation of (2-1) in row of the order of the word . The processing that use n equations in turn is called a *turn processing*. The value of the state word is decided, and a state becomes 1 when all the state of that starting-point are 1. The state of the word which is not a starting-point isn't concerned for the state of the end-point of that equation. The matter that a turn processing is repeated is called a *iteration processing*. Even if it is repeated, word that is not decided truth becomes the state (-1) of falsehood, or remains null. When the above thing is expressed in the directed graph, it can be said,

(1) Value of node (word) isn't decided if a node wasn't contained in the closed loop and if either starting-point is a falsehood when we searched a starting-point one after another from that node. Value of node (word) isn't decided if the value of the word didn't meet user's necessary conditions and a falsehood was taken even if the value is calculated successfully.

(2) The value of the node is decided if a node wasn't contained in the closed loop and if all the starting-point are truth when we searched a starting-point one after another from that node.

(3) The value of the node is remained null when a node is contained in the closed loop and when the starting-point of the node doesn't exist in system.

(Example 2) The state of x_3, x_4, x_5, x_6 and x_7 depends on the state of x_1 and x_2 which are final starting-points in case of an example 1 of the figure 2. When x_1 which is final starting-points come truth, x_5 is truth. When x_1 and x_2 which are final starting-points is truth, x_4 and x_5 come truth. x_6, x_7 are in the loop, and so remain null.

4.Operation with the adjoin matrix F and the State vector X

An arithmetic operator is defined so that it may become the same value that it is calculated with doing a iteration processing with the adjoin matrix F and the state vector X.

In other words, multiplication with F and X is,

$$Y = FX \quad \text{where } X, Y \text{ are column vector} \quad (4-1)$$

The element y_i of Y takes the inside product the i -th row vector of F and X as follows

$$y_i = f_{i1} \cdot x_1 + f_{i2} \cdot x_2 + \cdots + f_{in} \cdot x_n \quad (4-2)$$

A multiplication arithmetic operator (\cdot) is defined as follows.

Axiom 1

$$\begin{array}{lll} 0 \cdot (-1) = 0 & 0 \cdot \text{null} = 1 & 0 \cdot 1 = 1 \\ 1 \cdot (-1) = 0 & 1 \cdot \text{null} = 1 & 1 \cdot 1 = 1 \end{array} \quad (4-3)$$

An addition arithmetic operator ($+$) is defined as follows.

Axiom 2

We think about ($+$) with $x_i = \sum_j f_{ij} \cdot x_j$

$$\begin{array}{lll} (-1) + (-1) = (-1) & (-1) + \text{null} = (-1) & (-1) + (1) = (-1) \\ \text{null} + (-1) = (-1) & \text{null} + \text{null} = \text{null} & \text{null} + (1) = \text{null} \\ (1) + (-1) = (-1) & (1) + \text{null} = \text{null} & (1) + (1) = (1) \text{ or } (-1) \end{array} \quad (4-4)$$

As for the meaning of the addition arithmetic operator ($+$), the right side is (-1) if there is even only one (-1) , the right side is null if there isn't (-1) in the left side and there is even only one null . If all of the left sides are 1, the right side is 1 or, (-1) .

We prove the following theorem by using the axiom 1,2.

Theorem 1

The state of the end point becomes 1 (truth) when all the state of the starting-point is 1 (truth).

(Proof)

If a starting-point is truth, it becomes $1 \cdot 1 = 1$ by the axiom 1. As for the case that it is not a starting-point, it becomes $0 \cdot * = 1$ ($*$ is either of three state). Therefore, when addition is calculated, it becomes 1 (truth) or (-1) (falsehood) by the axiom 2. There is no condition except for this by the axiom 2.

Theorem 2

The state of the end-point becomes (-1) (falsehood) either when the state of the starting-point is (-1) (falsehood) or when all the starting-point are 1 (truth).

(Proof)

If either state of the starting-point is (-1) , there is a case in which becomes $1 \cdot (-1) = (-1)$ by the axiom 1. Therefore, a result becomes (-1) (falsehood) by the axiom 2. It is proved by the theorem 1 when all starting-point are 1 (truth).

Theorem 3

The state of the end point becomes (null) (It is undecided.) when the state of one and more starting-point is null and the state of the rest of the starting-point is (1) (truth).

(Proof)

If the state of one and more starting-point is null, the state of the end point becomes null by the axiom 1. If the state of one and more starting-point is null, the state of the end point becomes null by the axiom 1.

The turn processing is to process $Y=FX$. The iteration processing is to process.

$$Y = FX$$

$$Y' = FY = F^2X$$

$$Y'' = FY' = F^3X$$

M

repeatedly. The state of the end-point changes in 1 or (-1) from null by the iteration processing which repeats the turn processing. As for that state, the state of the end-point doesn't change even if the thing of 1 (truth) and (-1) (falsehood) carries out a iteration process. The state of the end point is left null even if we repeat it many times when a closed loop exists in the graph and an end point is in the closed loop or this system (world) doesn't have a starting-point when we show it by directed graphs.

5. The structure of the program of every word

Under the consideration until now, the program to calculate the equation (2-1) for every state word is as follows,

(1) To judge whether to carry out the equation

(2) To carry out the equation (2-1) when the state of word α is null. We carry out the equation (2-1), and memorize it in the temporary area. A temporary area is necessary.

(3) To judge whether value was decided or not as a result of carrying out the equation (2-1). Value is decided when all starting-point are truth. But, even if all the starting-point is truth, word may be made falsehood on the user's requirement condition.

(4) To record a result in the normal area when value is decided. A normal area is necessary.

(5) To judge whether state is falsehood. When word is made falsehood with user necessary conditions or when a starting-point is falsehood, state is falsehood.

(6) The flag which shows whether value was decided or not is necessary for every word as a process of the falsehood. (Or, we memorize the value which means that the state is falsehood in the normal area.)

(7) We add 1 to the comeback counter to judge whether we carry out iteration process or not. Comeback counter is necessary [4].

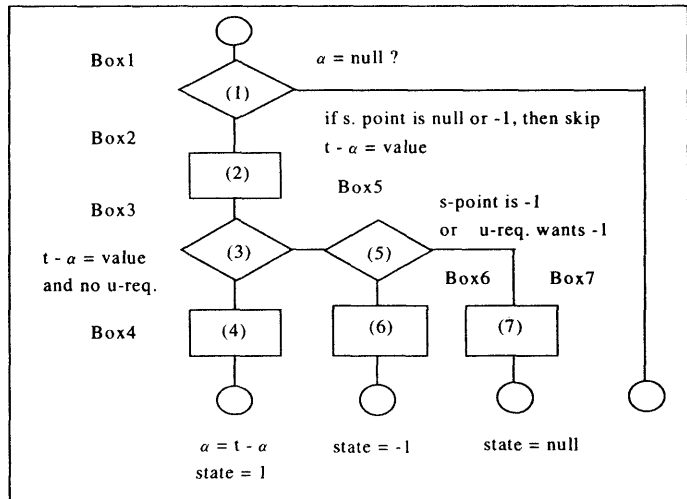


Fig.5-1 The program structure of a word (α)

6. The program to control the whole of the system and that structure

A program to control the whole system (world) is called *the tense control function* [4]. The control program to do iteration processing properly is put in this program. If a state of starting-point is falsehood, the state of end-point surely becomes falsehood. A state of end-point becomes truth when falsehood is taken with user necessary conditions even if all the state of starting-point is truth. Therefore, we have only to judge the matter whether the

starting-point of null is contained in the equation to judge whether a repetition (comeback) does or not. If it has even one starting-point of null, the iteration processing is done. However, when we express it by directed graphs, a state of starting-point doesn't change from null when a closed loop is contained. And, it is finished null when this system (world) doesn't have a starting-point. Therefore, the iteration process isn't finished and we go into the permanent loop. There is a comeback counter in the tense control function program (Φ) which controls the whole of the program of every word. Every time that word becomes null, each word program adds 1 to the comeback counter. As a condition to pass a series of iteration process, we set up that the value of this counter is the same as the last value. We say this state as "There is no change in a state."

(Example 3)

```

Ctr = 0
Do
  PCtr = Ctr
  Ctr=0
  L(x)
  L(y)
  L(z)
Loop Until Ctr = PCtr

```

7. The nature of the adjoin matrix

F is an adjoin matrix, X is a state vector (what showed the state of the word of the system (world) in the vertical vector). n is the repetition number of times. Using the equation (4- 1), "There is no change in a state." is shown as follows,

$$F^n X = F^{(n-1)} X \quad (7- 1)$$

In other words, one of these equations is truth.

$$(F - I) = 0 \quad (7- 2)$$

$$F^{(n-1)} = 0 \quad (7- 3)$$

$$X = 0 \quad (7- 4)$$

When an equation (7-2) is concluded, the eigen value of F is on the perimeter of the radius 1. At this time, a closed loop occurs on the directed graph, and then the value of null rotates on the perimeter of the radius 1. F is a nilpotent matrix when an equation (7-3) is concluded. As for this matrix, if F is multiplied k times, then all clauses become 0 where k is the size of the biggest chain of the partial order set (F, \leq). $k \leq (n-1)$. We say B as the chain when the subset B which is not the empty of the partial order set (F and \leq) is a total order under \leq .

As for the condition that F is a nilpotent matrix, all the characteristic polynomial expressions are t^n and 0 as for the eigen value. Therefore, if a directed graph doesn't contain a closed loop, we can arrange F again for lower (or, upper) triangular matrix of the opposite angle clause 0.

8. How to line up the word Iteration process unnecessary

The iteration process can be avoided by lining up a word in turn process so that an adjoining matrix may become a lower triangular matrix when it is shown in the directed graph with no relations between the starting-point and the end-point in the closed loop as shown in Fig. 8-1.

. We call it a topological sort that we arrange the data in one line from work to do first most to work to do at the end which partial order relations are given. An answer isn't always decided uniquely because they are data related to the partial order. We can carry out a topological sort by doing a depth-first search to the directed graph, and picking up a node when we are returned. How to line up the word of the turn process has only to line up a word in this method.

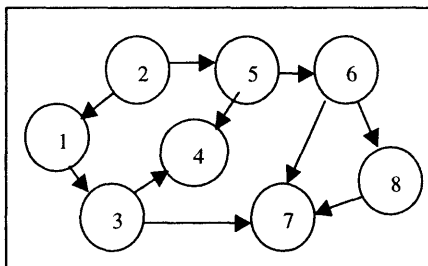


Fig. 8-1 Directed graph on example

(Example 4)

Adjoin matrix of this directed graph is,

$$F = \begin{matrix} & \begin{matrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \end{matrix} \\ \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \end{matrix} & \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} \end{matrix}, X = \begin{bmatrix} \text{null} \\ 1 \\ \text{nul} \\ \text{nul} \\ \text{nul} \\ \text{nul} \\ \text{nul} \\ \text{nul} \end{bmatrix} \quad (8-1)$$

The result of topological sort is (1,2,3,4,5,6,7,8)? (2,5,6,8,1,3,7,4).

We have sorted F topologically, then F'(sorted F) is

$$F' = \begin{matrix} & \begin{matrix} 2 & 5 & 6 & 8 & 1 & 3 & 7 & 4 \end{matrix} \\ \begin{matrix} 2 \\ 5 \\ 6 \\ 8 \\ 1 \\ 3 \\ 7 \\ 4 \end{matrix} & \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} \end{matrix} \quad (8-2)$$

The structure of a program to make in every word and the structure of the tense control function program with which we control that became clear in the above. And, it was proved that they worked mathematically properly. We think about the structure in which the number of words is large based on these fundamental ways of thinking from here.

9. The division of the system (world)

We thought about all words in one system (i.e., world) so far. We can divide words into the input words which a person inputs that value to or which value is decided as with other systems and the state words which form value in this system from the starting-point. And, the output word to output it to the outside of the system from the state word including the input word is necessary. We divide this formation process into three of the input process, the formation process and the output process.

We call the structure of these three processes as the basic structure, and call each process a palette.

10. The mathematical principle of the palette division

An adjoining matrix $F1$ after the palette division is a unit matrix. Therefore, it doesn't influence an iteration process. An adjoining matrix $F2$ is the matrix of $n \times n$, and equivalent to F of the equation (2-3). $F3$ is a $m \times n$ matrix in which only one 1 appears in each line, and doesn't influence a iteration process as well as the unit matrix. Therefore, the theorem concluded with F is concluded as it is.

The word inputted at the same time, and the word outputted at the same time are handled respectively as a group.

11. The division of the basic structure

The words inputted at the same time, and the words outputted at the same time are handled respectively as a group. This unity is caused with the logical record (logical body). The group of the words outputted at the same time is assigned to one basic structure. By using this concept,

(1) The range of the iteration process becomes small.

(2) One of the basic structures can be used repeatedly, and sequential read and algorithm such as summation can be realised by this in the series of process.

The divided basic structure is carried out in the fixed order one after another. The matter that it transits from the basic structure to the basic structure is called a *link*. When the state of all the output words of the basic structure is not null and when output was completed, then that basic structure links to another basic structure. If the state of one starting-point is null, and if the state of the output word is null, next basic structure is searched by the depth-first search and linked to the basic structure of which the null starting-point is an end-point. If the output word of null disappears, it returns to the basic structure in which null appeared in first, and then this process is done again. This process is repeated until whole change in a state of the system disappears. The diagram in which the state of the link of the basic structure is illustrated is called a process route diagram, or PRD [5]. Now, we have thought that the word, which we formed once, keeps existing, and when it is necessary, we can use it as a starting-point from any basic structure. When we use the same basic structure several times, we must use the word domain of the basic structure several times. So, we make a temporal logical record (logic body) on the tense control function program, and output it there for a while.

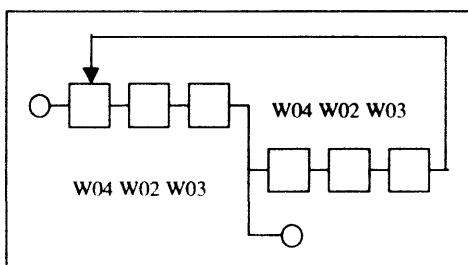


Fig. 11-1 Process route diagram

The palette domain of that basic structure is cleared with null once, and we input the value of the necessary word to that basic structure as an input word again. The algorithm that we use the same basic structure for several times such as a sequential read and a summation algorithm can be realised by this structure.

12. The mathematical principle of the division of the basic structure

When we divide basic structure, the relation occurs that state word of a certain basic structure S_i is used as a starting-point word of other basic structure S_j . We think about the directed graph that basic structure is a node. When the word of the basic structure S_j uses the state word of the basic structure S_i as a starting-point, we define that there is a branch to the node S_j from the node S_i . When a closed loop is contained in the directed graph which we showed as the relations which we used a branch for as a starting-point, we combine a closed loop with one basic structure (i.e., node). Until a closed loop disappears, we continue this operation. The basic structure left by the state that a closed loop disappears is a total order.

(Example 5)

Next, we think about the directed graph that has five nodes which is shown in Fig.12-2.

A node 2 and a node 4 are combined because a closed loop is formed, shown in Figure12-3. An combined node 24, and a node 3 are combined again because a closed loop is formed which is shown in Figure12-4. The above operation means that if we think in the adjoining matrix of the word being used with each basic structure, we are looking for the following block lower triangular matrix.

$$F = \begin{bmatrix} F1 & 0 & 0 \\ F1 \rightarrow F234 & F234 & 0 \\ F1 \rightarrow F5 & F234 \rightarrow F5 & F5 \end{bmatrix}$$

where

F: adjoin matrix of whole system

F1: adjoin matrix of S_1

F234: adjoin matrix of $S_{2,3,4}$

F5: adjoin matrix of S_5

$F1 \rightarrow F234$: adjoin matrix from structure1 to $S_{2,3,4}$

$F1 \rightarrow F5$: adjoin matrix from S_1 to S_5

$F234 \rightarrow F5$: adjoin matrix from $S_{2,3,4}$ to S_5

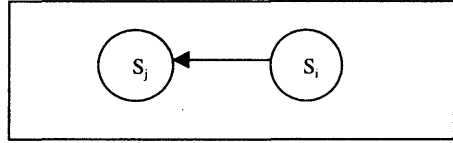


Fig. 12-1 Directed graph of basic structure

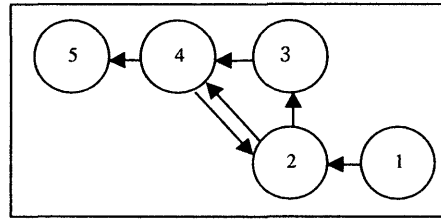


Fig. 12-2 Directed Graph of example

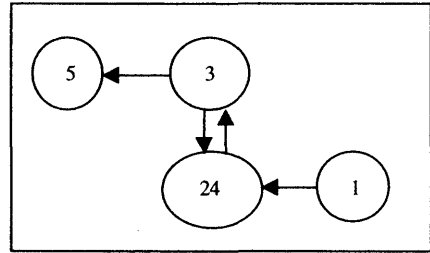


Fig. 12-3 Combined directed Graph

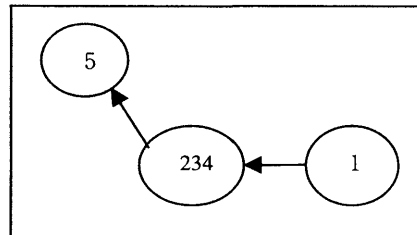


Fig. 12-4 Combined directed graph (2)

13. The tense control function and its mathematical Principle

The meaning of tense control function and each action vector on it is explained from the position of a-word-based-program, and the mathematical correctness is verified.

1) Division of a base structure, and the routing vector

A base structure is divided for every output defined. When two or more signification vectors have the same conditions, these groups are the defined logical unit.

(Example 6)

In the case of an input file, one record is the defined logical unit. One record is the group of one or more keyword(s) (sequence is defined) and words. In the case of a screen, it is the group of the word inputted by an input command (function key). Therefore, among those words in one defined logical unit, it does not have the logical relation between the starting-point and the end point. This is the same concept as that of the normalisation of file generally said. The routing vector is an action vector which directs W04 performed next after execution of W03 in a certain base structure. A routing vector determines the sequence of execution. However, it is verified by the predicate structure that a process result is not influenced by the sequence of execution as mentioned above. In normal case, all base structures are linked by the depth-first search. The execution conditions of the routing vector are equivalent to the truth condition of the starting-point at which W03 signification vector of the base structure was common. It is equivalent that the executive condition of the routing vector is satisfied with the truth condition of more than one signification vector being satisfied. That is, all W03 signification vectors fulfil the condition. According to this fact, instead of carrying out conditional judgement by performing each signification vector, it is possible to increase the efficiency of a process by carrying out judgement in advance whether we perform a base structure or not. A process route diagram is used as a tool of the design for catching the requirements of user correctly. For this purpose, the ease of catching of requirements serves as a measure of the merit of a process route diagram. The base structure which outputs the last result is placed first, and a base structure required in order to obtain the result is linked one after another. This method of link of a base structure tends to catch the requirements of user. This is called end-point-oriented approach. On the other hand, the process efficiency at the time of execution is greatly influenced by specification of a route. Process efficiency is good when it processes toward the last end-point from the last starting-point, generally. For this reason, a route is sometimes corrected so that the optimal process speed may be obtained after a requirement design. This efficiency can be quantified with the processing time, the number of steps, etc. when changing a route.

2) The action vector of structure

A part of program is sometimes used repeatedly. An input word needs to be repeated in order to realise this. For this reason, the action vector of structure clears the area of a word by null if needed.

(Example 7)

In the case of the case where key is inputted from a screen and 1 record of a file is displayed on a screen, then, the other value of key is inputted and another 1 record is displayed. The base structure of a screen and a file realises this process by clearing the area of a word by null, if it is displayed or outputted.

(Example 8)

The value of a certain word of each record of a file is totalled from Top to EOF. In this example, in order to input into the next process, the last value of the output of that base structure is transmitted to a boundary area which is shown M in Fig.13-1. Then, a word area is cleared by null. The mathematical correctness of these processes is proved using the theorem for a verification of a predicate structure.

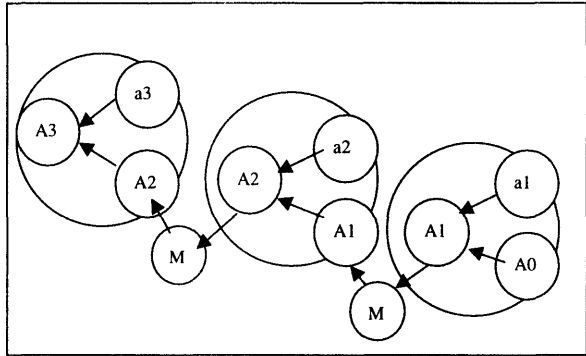


Fig. 13-1 Directed Graph of Summation

3) The action vector for input

Transmission of the value to the input buffer from an input unit is the role of the action vector of input. It is transmitted to W02 signification vector after carrying out check of a type. By this, the signification which people judged from the first is materialised on a computer. In order for an action vector to identify equipment and to specify record, the output to equipment is required in spite of the action vector of input. Moreover, the equipment side may have memorized the position of pointer like sequential read, and in such case, a control of pointer is required.

4) The action vector of output

The signification vector of W04 makes a value to W04 word domain. All processes that connect the value of W04 with an output unit are processes of an action vector. Usually, the flag showing completion, a success, etc. is returned from an equipment side. The action vector of input and output are the interface between the logical and mathematical world of words, and a man and an external device, and a verification in the logical and mathematical world of a word is another field.

14. Conclusion

In the present work, we have built the mathematical structure model of a word-unit-based program defining states of word. The state of word configures a complete partial order, which have null as a minimum element of word. Using a directed graph and its adjoint matrix, this structure proved to be mathematically proper. Furthermore, we have explained the correctness of the general idea of the palette and the basic structure, which are necessary to make a practical program. We have discussed the structure to apply algorithm such as sequential read, summation in this paper. These general ideas are used with the system development methodology of Lyee, which gets many results that have been already practical in many industrial applications.

Interface and communication with the human user, the outside device, are discussed. This warrants actual work on the modelling of a word-unit-based program, for such type of system.

15. Acknowledgement

While we are carrying out this research, we got the guidance and the advice of Mr. Negoro Fumio who is the inventor of Lyee. Fruitful discussions with Mr. Sigeji Hashimoto are greatly appreciated.

References

- [1] Osamu Arai, Hamid Fujita, The Mathematical Structure model of a Word-unit-based Program, SSRR 2002s computer & internet Conference , L'Aquila , Italy, July ,29-August 4 2002
- [2] Glynn Winskel, The Formal Semantics of Programming Languages, The MIT Press, 1993
- [3] Hanne Riis Nielson, Flemming Nielson, SEMMANTICS WITH APPLICATIONS A Formal Introduction, John Wiley & Sons,1992
- [4] Fumio Negoro, "Principle of Lyee Software", Proceedings of 2000 International Conference on Information Society in the 21st Century (IS2000), Aizu, Japan, November 5-8, 2000, pp441-446.
- [5] Fumio Negoro, "Intent Operationalisation for Source Code Generation", Proceedings of the 5th World Multi-Conference on Systemics, Cybernetics and Informatics (SCI2001), Orlando, USA, July 22-25, 2001.

Oamu Arai

He is working as project manager in Catena Co-operation, http://www.catena.co.jp/english/eng_index.htm/ , and he is project manager for many business projects, in Japan. He is currently also a member of the graduate studies of Iwate Prefectural University, Graduate studies of software and information Science for Doctor degree under the supervision of Prof. H. Fujita.

A Top-Down Approach to Identifying and Defining Words for Lyee Using Condition Data Flow Diagrams

Shaoying LIU

Department of Computer Science

Faculty of Computer and Information Sciences

Hosei University, Tokyo, Japan

Email: sliu@k.hosei.ac.jp

URL: <http://www.k.hosei.ac.jp/~sliu/>

Abstract. We present a top-down approach to identifying and defining words for the Lyee system using the visual formalism known as Condition Data Flow Diagram used in the SOFL (Structured Object-Oriented Formal Language) formal engineering method. The proposed technique can assist the analyst to identify effectively the necessary words as outputs of operations in a structured manner and to represent the relations among words in a visual formalism.

1 Introduction

The most important concern in writing a program for the Lyee (Governmental Methodology for Software ProviDenceE) System is centered on the definitions of *words*. A word represents a data element and is usually defined in terms of other words or constants (concrete values of certain types) [1]. From the Lyee's point of view, a software is an operation producing an output word based on input words or constants. Once all the necessary words are defined precisely and necessary inputs for the definitions are given, regardless of the order in which the definitions of words are organized, the Lyee system can automatically work out the concrete value for the output word. It is claimed that an advantage of the Lyee paradigm of programming over the traditional programming paradigms (e.g., imperative programming) is to allow the programmer or analyst to concentrate on the definitions of data elements (words in Lyee) without the need of considering how they are organized [1].

However, several problems need to be addressed before a *correct* Lyee program, a *collection of definitions of words*, can be constructed. The first one is how to know what words for a specific system need to be defined; whether the identified words are complete; and whether those definitions are consistent. The second problem is how to represent the definitions so that they will be easily understood by both the customer and the analyst. This point is important especially when dealing with large-scale systems. Another problem is how to find out appropriate expressions for defining the words. For example, suppose we know that word a_1 is defined in terms of words a_2 and a_3 , represented formally as

$$a_1 = E(a_2, a_3)$$

but have no idea of how a_2 and a_3 can be used to define a_1 at the moment. The details of E is actually expected to be clarified as the analysis of the user requirements proceeds.

To address these problems, a systematic method and comprehensible notation are helpful. In this paper we adopt the visual formalism known as Condition Data Flow Diagram, or CDFD for short, as a method and notation for identifying and defining words in a top-down and modular manner. As the brief introduction to this visual formalism in section 2 reveals, CDFD is a formalization of traditional Data Flow Diagram by integrating it with Petri nets and pre-postcondition notation. It allows one to describe comprehensively all the related definitions of a word and to decompose an abstract definition of a word into detailed definition given in terms of other lower level words. Since the entire architecture of the word definitions can be organized as a hierarchy of CDFDs, the modularity of the entire word definitions can be achieved, which will help verification and evolution of the definitions. To make necessary distinction between CDFD representations and Lyee style definitions of words, we call the word definitions in CDFDs *specification* while the original Lyee style word definitions *program*.

The rest of the paper is organized as follows. Section 2 gives a brief introduction to CDFDs while section 3 explains how they can be used to define words in a hierarchical manner. In section 4 I elaborate how data abstraction can help achieve high levels of functional abstraction before the system is implemented in Lyee, and section 5 gives an example to show the application of CDFDs leading to Lyee programs. Finally, in section 6 I give conclusions and outline the future research.

2 Brief Introduction to CDFD

CDFD is a graphical notation for modeling system functions by defining data flows among processes. It is part of the SOFL (*Structured Object-Oriented Formal Language*) specification language, usually used for describing the architecture of specification. It was designed initially by integrating Data Flow Diagrams (DFD), Petri nets, and the formal specification language VDM-SL (*Vienna Development Method - Specification Language*) in 1992 as the result of my doctoral research project, and extended later to include Object-Oriented features based on my research efforts in collaboration with many international researchers over last ten years. SOFL has been applied to system modeling and design in both industrial and research projects [2][3].

A CDFD is usually composed of three kinds of components: processes, data flows, and data stores, as depicted by Figure 1. A process, represented graphically by a box, models a transformation from its input to output. For example, process P transforms input data flows a and b and generates two output data flows c and d , while process Q takes data flow c and produces either f or g but not both (this exclusive-or relationship between f and g is indicated by the short horizontal line between them). A data flow indicates that a data element moving from one process to another. For example, data flow c shows that a data element bound to c moves from process P to Q while d represents a data flow from P to W . A data store represents data depository (like a file or database), which can be accessed or updated by processes during their executions. For instance, data store $s1$, numbered 1, is updated by process P , indicated by the directed line from P to $s1$, whereas store $s2$ is read by both process Q and W , denoted by the directed lines from $s2$ to both Q and W .

A CDFD is different from a traditional DFD like Yourdon's [4] in several ways. Firstly, A CDFD has an operational semantics [5] like Petri nets [6] whereas a DFD has

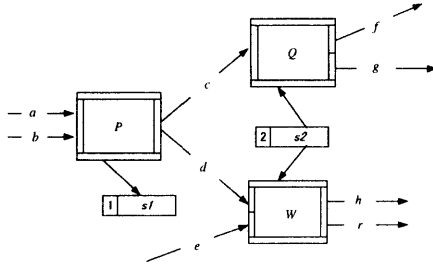


Figure 1: An example of CDFD

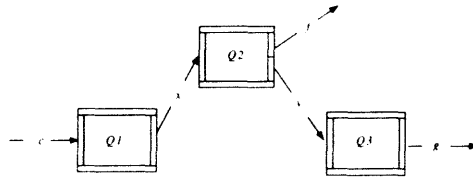
none. Take the CDFD in Figure 1 as an example. When both data flows a and b are available, process P is enabled and will be executed once its output data flows become unavailable. The execution will result in the generation of data flows c and d , which move to process Q and W , respectively. Then process Q executes and either f or g is produced. In the meanwhile, process W may also execute to generate both h and r . The executions of process Q and W can be performed in parallel. Secondly, the functionality of each process is defined with a pair of pre and postconditions, describing the condition that must be met by input data flows before the execution of the process and the condition that must be met by output data flows after the execution, respectively, whereas there is the lack of such a formal definition in DFDs. For example, we can define process P in an abstract form as follows:

$$P(a, b; c, d; s1) : [a > 0 \wedge b > 0, c = a + b \wedge d > a * b \wedge s1 = \sim s1 + a * b]$$

Where $a > 0 \wedge b > 0$ is the precondition; $c = a + b \wedge d > a * b \wedge s1 = \sim s1 + a * b$ is the postcondition of process P ; and $\sim s1$ denotes the initial value of variable $s1$ before the execution of P .

Note that the output data flows c and d as well as store $s1$ are defined based on the input data flows a and b in the postcondition, but not necessarily in a deterministic manner. For example, data flow c is defined in a deterministic manner as $c = a + b$ (c is equal to the sum of a and b) whereas data flow d is defined in a non-deterministic manner as $d > a * b$ (d is greater than the product of a and b). Finally, the relationships between either input data flows or output data flows of a process are precisely defined in CDFDs (e.g., data flows a and b are both required for execution of process P whereas only one of data flows d and e is necessary for execution of process W), but such relationships in DFDs are ambiguous at the diagram level.

Processes in a CDFD can be decomposed into the next lower level CDFDs if it is necessary to describe in detail how the inputs of the processes are transformed into their outputs. For example, assume that defining the functionality of process Q needs some intermediate data flows to bridge its input data flow c and output data flow f or g . In this case, it is reasonable to decompose Q into a lower level CDFD, as shown in Figure 2. To ensure the structural consistency between the decomposition and process Q , the input data flow to and the output data flows from the entire decomposed CDFD must be the same as those of process Q , respectively. Semantically, the decomposition of process Q refines the specification of Q given in the form of pre and postconditions.

Figure 2: A decomposition of process Q

That is, eventually process Q can be replaced by its decomposition in the CDFD of Figure 1.

Since the aim of this paper is to adopt CDFDs for defining words in Lyee programs, I do not introduce the other constructs associated with CDFDs in the SOFL specification language. Readers who are interested in the detail of SOFL can refer to our previous publications on SOFL [7][8][9][10].

3 Defining words using CDFDs

To help identify and define necessary words for a specific software system, I suggest that top-down approach be taken based on CDFDs. Since programming using Lyee starts with identifying and defining the output words of the entire system (sometimes it may be a subsystem of a large one) in terms of other words, it is natural to use a process for the abstraction of such a definition whose output data flows denote the desired output words and input data flows represent the other necessary words in the definition. If the relation between the output words and input words cannot be defined deterministically for some reason (e.g., lacking necessary details), the process can be decomposed into a lower level CDFD so that the output words can be defined based on its input words through some intermediate words. This way will give the analyst opportunities to identify and define additional words necessary to the system, such as the intermediate words involved in the decompositions of high level processes.

3.1 Liner definitions

By a liner definition of a word I mean that the word is defined based on other words without involving choices and recursive definitions. A simple example may help us explain how such a definition can be represented by a CDFD. Suppose we define words a , b , c , d , and f as follows:

$$\begin{aligned} a &= E_1(b, c) \\ b &= E_2(d, f) \\ c &= E_3(4, 10) \\ d &= 20 \\ f &= 50 \end{aligned} \quad (1)$$

and we do not know exactly how E_1 , E_2 , and E_3 should be defined at the moment. These definitions are represented by the CDFD in Figure 3.

The processes involved in this CDFD are defined as follows:

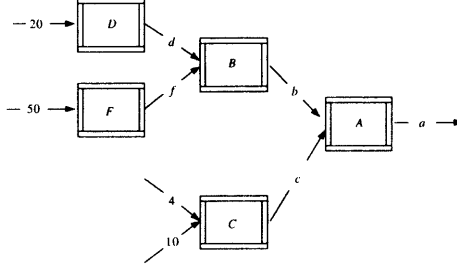


Figure 3: A CDFD representing the definitions of the words

$$\begin{aligned}
 A(b, c; a;) &: [true, a = E_1(b, c)] \\
 B(d, f; b;) &: [true, b = E_2(d, f)] \\
 C(4, 10; c;) &: [true, c = E_3(4, 10)] \\
 D(20; d;) &: [true, d = 20] \\
 F(50; f;) &: [true, f = 50]
 \end{aligned}$$

The precondition of each process, such as A , means that there is no specific constraint on its input data flows (words in this particular case). Since we may not be able to build some expressions for defining the output words, such as E_1 and E_2 , we need to decompose them into the next lower level CDFDs, respectively, to figure out how the expressions can be defined in terms of some other lower level words. For example, suppose process A is decomposed into the CDFD in Figure 4. Thus, the definition of word a given previously will become:

$$\begin{aligned}
 a &= E_1^1(q3) \\
 q3 &= E_1^2(q1, q2) \\
 q1 &= E_1^3(b) \\
 q2 &= E_1^4(c)
 \end{aligned}$$

Where $E_1^i (i = 1..4)$ denote expressions.

By replacing the definition of word a in (1), we update the definitions of the words to produce the following list:

$$\begin{aligned}
 a &= E_1^1(q3) \\
 q3 &= E_1^2(q1, q2) \\
 q1 &= E_1^3(b) \\
 q2 &= E_1^4(c) \\
 b &= E_2(d, f) \\
 c &= E_3(4, 10) \\
 d &= 20 \\
 f &= 50
 \end{aligned}$$

Of course, the decomposition of high level processes defining words can continue, if necessary, until all the words are defined in a deterministic manner. Note that the diagram of CDFD gives a two dimension representation of the words; it clearly shows which words are used to define which other words, while the formal definition of the related processes using pre and postconditions gives a precise formula for defining words.

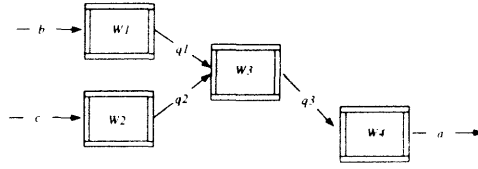
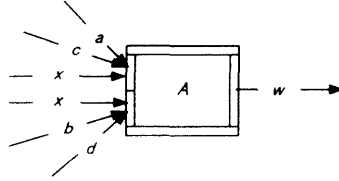
Figure 4: The decomposition of process A 

Figure 5: A graphical representation of choice construct

Therefore, the diagram and the process specifications are complementary in defining words.

3.2 Choice

A choice of executing different statements for different results is a common construct in many programming languages, but in Lyee such a choice is reflected when the same word is defined using different other words and expressions (computation formula). It seems that unlike normal programming languages, the condition for deciding which definition of the same word will be used to compute its value is not given clearly; instead, the decision will be made based on the availability of the constituent words of the word in its definition.

For example, suppose word w is defined as a choice as follows:

- (1) $w = a * x + c$
- (2) $w = b * x + d$

In (1) w is defined in terms of the words: a , x , and c ; but in (2) w is defined using: b , x , and d . Such definitions can be represented graphically as the process in Figure 5. Originally this process means that when either a , c , and x or d , b , and x are available, process A will execute and will generate w . In other words, which group of the data flows are used to generate w will depend on the availability of all the data flows in that group. This semantics is just well-suited for representing the definition of w as a choice. The formal textual specification of process A is:

$$A(a, c, x \mid x, b, d; w;): [true, w = a * x + c \vee w = b * x + d]$$

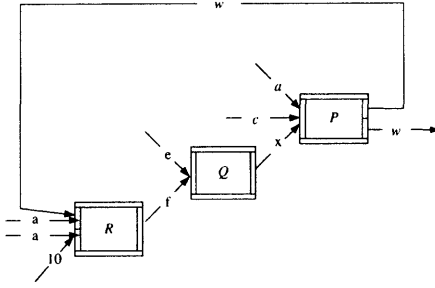


Figure 6: A CDFD for the recursive definition

3.3 Recursion

Another possible structure in Lyee programs is recursive definitions of words. Consider the definition of word w below as an example.

$$\begin{aligned} w &= a * x + c \\ x &= e + f \\ f &= a * w \\ f &= a + 10 \end{aligned}$$

Word w is defined using a , x , and c ; x is defined in terms of e and f ; and f is defined as a choice: either using a and w or a and 10. Thus, w is possibly defined recursively, that is, within the definition of w , w is used to define another word. Such a recursive definition is described by the CDFD in Figure 6.

4 Data abstraction

In the SOFL specification language that uses CDFDs for descriptions of specification architecture, not only are data items of basic types, such as integers, real numbers, and characters, but several compound data types can also be used to define data flows. Thus, it enables us to achieve high levels of abstraction.

In this section I give several examples to explain how the compound data types available in SOFL can be used to model complicated words.

Suppose a word is expected to represent a collection of elements or values, it does not seem to have an easy way to express it in Lyee. But this can be done easily using a value of a set type in SOFL.

Assume word w is used to hold such a compound data item. Then we declare it as:

$w : \text{set of int};$

Thus, w may take the following values, each being a set of integers.

- (1) $w = \{3, 6, 9, 12\}$
- (2) $w = \{5, 15, 25\}$
- (3) $w = \{10, 20, 30, 40, 50\}$

If duplication of elements is allowed in a word of set type or the order of the occurrences of elements is significant in determining the feature of the word, we can use another type, known as sequence, to define the word, for example,

$w : seq\ of\ int;$

Thus, w may take the following values:

- (1) $w = [3, 6, 9, 6, 12, 9]$
- (2) $w = [5, 15, 25, 35]$
- (3) $w = [10, 20, 20, 10, 30]$

If word w is expected to describe a composite value that has several fields, then we declare it as:

$w : composed\ of$
 $f_1 : TY_1$
 $f_2 : TY_2$
 \dots
 $f_n : TY_n$
 $end;$

With this declaration the fields f_1, f_2, \dots, f_n of word w will be accessed or updated, depending on the operations applied to them. For example, w can take the following composite value:

$w.f_1 = 10$
 $w.f_2 = 20$
 $w.f_3 = 30$

Where $n = 3$.

In addition to set, sequence, and composite types, there are also other types in SOFL, such as map types and union types. But since they are too complicated for abstraction of Lyee programs, I do not elaborate them here. The details of these types are given in our previous publications [7][11].

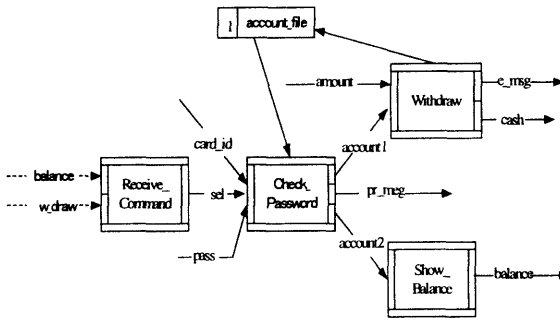
5 Example

I give an example to show the process of first using CDFDs to model the user requirements and then transforming it into Lyee-like word definitions. The system we deal with is a *Cash Dispenser*.

5.1 System description

Suppose the cash dispenser is required to have the following functions:

- (1) Provide the buttons for *showing the balance* of, and *withdraw money* from, the account.
- (2) Insert a cashcard and supply a password.
- (3) If *showing the balance* is selected, the current balance is displayed.
- (4) If *withdraw* is selected, the requested amount of the money is withdrawn.



To model this system, we draw the CDFD in Figure 7. The overall output data flow, representing a word in Lyee’s term, of the system is either withdrawn *cash* or displayed account *balance*. In addition to these two outputs, error messages, as a kind of output, may also be possible, such as *e_meg* and *pr_meg* given in the CDFD.

The diagram conveys the user functional requirements and the dependency relations between the functions represented by the processes in the diagram. The selection of *balance*, denoting the command of *showing the balance* or *w_draw*, denoting the command of *withdraw*, is handled by the process *Receive_Command*. This process then generates a data flow *sel* to indicate which command has actually been selected, and passes this information to the process *Check_Password*. When the requested cash card *card_id* and password *pass* are provided, this process will check whether the provided account exists in the system account database *account_file*, and if so, whether the password *pass* is the same as that of the account. If these pieces of information are confirmed, the process *Check_Password* will pass the account information, denoted by *account1* or *account2*, to either the process *Withdraw* or *Show_Balance*, depending upon the value of data flow *sel*. If these pieces of information are, however, not confirmed, an error message *pr_meg* will be issued. The process *Withdraw* updates the *account_file* by reducing the *amount* from the current balance of the *account1* if the requested amount of money is less or equal to the current balance. However, if this is not the case, the process will generate an error message to show that the amount requested is invalid. The process *Show_Balance* takes the confirmed account denoted by *account2*, which is the same as *account1* in contents, and displays the current balance of the account, which is represented by the data flow *balance*.

In contrast with the informal functional specification given at the beginning of this section, the functional abstraction expressed by the CDFD is obviously more precise and comprehensible in modeling the dependency relations between processes. To completely define the CDFD, all the processes, data flows, and stores must be defined precisely in a proper manner. To achieve this goal, we adopt the *module* structure in SOFL to provide a formal specification of the CDFD. The module for the cash dispenser is given as follows:

```
module SYSTEM_Cash_Dispenser /* This module is the top level
                                module.*/
type
```

Account = composed of

account_no: nat
 password: nat
 balance: real
end

var

ext #account_file: set of Account; /* the account_file is an
 external store that exists
 independently of the
 cash dispenser. */

inv

forall[x: Account] | 1000 <= x.password <= 9999;
 /* The password of every account must
 be a natural number with four digits. */

behav CDFD_1; /* Assume the cash dispenser CDFD is
 numbered 1. */

process *Init()*

ext account_file

post account_file = ~account_file

end_process; /* The initialization process does nothing
 in updating the local store because there
 is no local store in the CDFD to initialize. */

process *Receive_Command*(balance: **sign** | w_draw: **sign**) sel: **bool**

post balance <> nil and sel = **true** or w_draw <> nil and sel = **false**

comment

This process recognizes the input command: show balance or withdraw cash. The output data flow sel is set to true if the command is showing balance; otherwise if the command is withdrawing cash, sel is set to false.

end_process;

process *Check_Password*(card_id: **nat**, sel: **bool**, pass: **nat**)

 account1: Account | pr_meg: **string** |

 account2: Account

ext rd account_file /*The type of this variable is omitted
 because this external variable has
 been declared in the var section. */

post sel = **false** and

 (**exists!**[x: account_file] |

 x.account_no = card_id and

 x.password = pass and

 account1 = x)

or

 sel = **true** and

 (**exists!**[x: account_file] |

 x.account_no = card_id and

```

        x.password = pass and
        account2 = x)
or
not (exists![x: account_file] |
        x.account_no = card_id and
        x.password = pass) and
        pr_meg = "Reenter your password or insert the correct card"
comment

```

If sel is false and the input card_id and pass are correct with respect to the exiting information in account_file, the account information is passed to the output account1. If sel is true and the input card_id and pass are correct, the account information is passed to the output account2. However, if neither the card_id nor pass is correct, a prompt message pr_meg is given.

```

end_process;

process Withdraw(amount: real, account1: Account)
    e_msg: string | cash: real
ext rw account_file
pre account1 inset account_file
    /*input account1 must exist in the account_file*/
post (exists[x: account_file] |
        x = account1 and
        x.balance >= amount and
        cash = amount)

    and
    account_file = union(diff(~account_file, {account1}),
        {modify(account1, balance -> account1.balance - amount)})
or
not exists[x: account_file] |
    x = account1 and
    x.balance >= amount and
    e_meg = "The amount is too big")

comment

```

The required precondition is that input account1 must belong to the account_file. If the request amount to withdraw is smaller than the balance of the account, the cash will be withdrawn. On the other hand, if the request amount is bigger than the balance of the account, an error message "The amount is too big" will be issued.

```

end_process;

process Show_Balance(account2: Account)
    balance: real
post balance = account2.balance;
end_process;
end_module;

```

Since there is no local store in the CDFD of this module, the initialization process *Init* does nothing in updating the local stores of the CDFD, as indicated by the postcondition of *Init*. Some relatively complicated process specifications are explained informally in the comment parts, such as processes *Receive_Command*, *Check_Password*, and *Withdraw*, but for the simple process like *Show_Balance* no comment is provided.

Notice that several operators defined in set types and composite types are used, such as **union**(), **diff**(), **modify**(), etc. Briefly speaking, the operation **union**(x, y) is the union of the two sets x and y ; **diff**(x, y) yields the set whose elements belong to x but not y ; and **modify**($x, f \rightarrow v1$) yields a new composite object from the given composite object x by replacing the value of its field f with $v1$.

5.2 Transformation

Suppose this specification is verified and validated to meet the user requirements. We then need to transform it into a Lyee program for possible execution by the Lyee system. Two issues given below need to be addressed for the transformation.

- Derivation of the structure of word definitions in terms of the word relations based on the CDFD.
- Generation of the computation formulas used in the definitions of words based on the formal specification of the processes involved in the CDFD.

These two issues are actually related closely with each other. If the specification of process is described implicitly, meaning that the relation between the input data flows and output data flows are written in a predicate, it is difficult to give a general rule for the derivation of the structure of word definitions and the associated computation formula. However, if the relation is described explicitly, meaning that the output data flows are defined with an expression, the structure of word definitions and the computation formula can be derived easily. Since most processes in the Cash Dispenser example are given in an implicit manner, their transformation into word definitions requires additional efforts from the programmer. Since there is no sufficient knowledge about how to deal with this kind of problem in Lyee, I would like to leave this topic for discussion at the workshop.

6 Conclusions and future research

I have presented a top-down approach to identifying and defining words for the Lyee system using the visual formalism known as Condition Data Flow Diagram used in the SOFL (Structured Object-Oriented Formal Language) formal engineering method. The proposed technique allows the analyst to identify effectively the necessary words as outputs of operations (processes) in a structured manner and to represent the relations among words in a visual formalism. Also, due to the availability of abstract data types in SOFL, such as set, sequence, and composite types, the functions of processes defining words can be expressed with a high level of abstraction, which allows the analyst to focus on the functions of the system before undertaking its implementation.

To make the proposed technique really useful in supporting Lyee programming, we need to provide effective techniques and tool support for transforming a CDFD into a set of word definitions. I am interested in the investigation of this issue in the future.

7 Acknowledgement

I would like to thank Prof. Hamid Fujita for his support and encouragement for this research.

References

- [1] The Institute of Computer Based Software Methodology and Technology, *Introduction to Lyee - Revolution by Automatic Software Development*, Distributed booklet, 2000.
- [2] Shaoying Liu, Masashi Asuka, Kiyotoshi Komaya, and Yasuaki Nakamura, "Applying SOFL to Specify A Railway Crossing Controller for Industry," in *Proceedings of 1998 IEEE Workshop on Industrial-strength Formal Specification Techniques (WIFT'98)*, Boca Raton, Florida USA, October 20-23 1998, IEEE Computer Society Press.
- [3] Shaoying Liu, Masaomi Shibata, and Ryuichi Sat, "Applying SOFL to Develop a University Information System," in *Proceedings of 1999 Asia-Pacific Software Engineering Conference (APSEC'99)*, Takamatsu, Japan, December 6-10 1999, pp. 404-411, IEEE Computer Society Press.
- [4] Edward Yourdon, *Modern Structured Analysis*, Prentice Hall International, Inc., 1989.
- [5] Chris Ho-Stuart and Shaoying Liu, "A Formal Operational Semantics for SOFL," in *Proceedings of the 1997 Asia-Pacific Software Engineering Conference*, Hong Kong, December 1997, pp. 52-61, IEEE Computer Society Press.
- [6] Wilfried Brauer, Grzegorz Rozenberg, and Arto Salomaa, *Petri Nets - An Introduction*, Springer-Verlag, Berlin Heidelberg, 1985.
- [7] Shaoying Liu, A. Jeff Offutt, Chris Ho-Stuart, Yong Sun, and Mitsuru Ohba, "SOFL: A Formal Engineering Methodology for Industrial Applications," *IEEE Transactions on Software Engineering*, vol. 24, no. 1, pp. 337-344, January 1998, Special Issue on Formal Methods.
- [8] Shaoying Liu, Masashi Asuka, Kiyotoshi Komaya, and Yasuaki Nakamura, "An Approach to Specifying and Verifying Safety-Critical Systems with Practical Formal Method SOFL," in *Proceedings of Fourth IEEE International Conference on Engineering of Complex Computer Systems (ICECCS'98)*, Monterey, California, USA, August 10-14 1998, pp. 100-114, IEEE Computer Society Press.
- [9] Shaoying Liu, "An Evolution Approach for Software Development Using SOFL Methodology," in *Proceedings of International Workshop on Principles of Software Evolution (submitted)*, 1998.
- [10] Shaoying Liu, "Formal Engineering Methods for Information Systems Development," in *Proceedings of 2nd International Conference on Information (INFORMATION2002)*, July.
- [11] Shaoying Liu, "Verifying Consistency and Validity of Formal Specifications by Testing," in *Proceedings of World Congress on Formal Methods in the Development of Computing Systems*, Jeannette M. Wing, Jim Woodcock, and Jim Davies, Eds., Toulouse, France, September 1999, Lecture Notes in Computer Science, pp. 896-914, Springer-Verlag.

A Linguistic Method Relevant for Lyee

Gregers KOCH

*Department of Computer Science, Copenhagen University
DIKU, Universitetsparken 1, DK-2100
Copenhagen, Denmark
Tel.: (+45) 35 32 14 00
Fax.: (+45) 35 32 14 01
Email: gregers@diku.dk*

Abstract

Here is presented and discussed some principles for extracting the semantic or informational content of texts formulated in natural language. More precisely, as a study of computational semantics and information science we describe a method of logical translation that seems to constitute a kind of semantic analysis, but it may also be considered a kind of information extraction. We discuss the translation from Dataflow Structures partly to parser programs and partly to informational content. The methods are independent of any particular semantic theory and seem to fit nicely with a variety of available semantic theories. We relate the method here to the Lyee Project. To illustrate the principles we discuss the detailed analysis of some carefully selected prototypical database queries formulated in simple English.

1 Introduction

Here is presented and discussed some principles for extracting the semantic or informational content of texts formulated in natural language. More precisely, as a study of computational semantics and information science we describe a couple of methods of logical translation that may be considered a kind of information extraction. We discuss the translation from dataflow structures partly to parser programs or logic grammars and partly to informational content. The methods are independent of any particular semantic theory and seem to fit nicely with a variety of available semantic theories.

Information is a concept of crucial importance in any conceivable scientific endeavour. Also the modeling of information or semantic representation is becoming more and more important, not least in information systems. Databases, knowledge bases as well as knowledge management systems are continually growing. Modeling helps to understand, explain, predict, and reason on information manipulated in the systems, and to understand the role and function of components of the systems. Modeling can be made with many different purposes in mind and at different levels. It can be made by emphasising the users' conceptual understanding. It can be made on a domain level on which the application domain is described, on an algorithmic level, or on a representational level. Here the interest is focused on modeling of information on a representational level to obtain sensible semantic representations and in particular on the flow of information between the vertices of a structure describing a natural language utterance.

We are in the habit of considering the syntactic phenomena, and especially those concerning parsing, as essentially well understood and highly computational. Quite the opposite seems to be the case with semantics. We shall argue that certain central semantic phenomena (here termed logico-semantic) can be equally well understood and computational. So, in this rather limited sense we may claim that the semantic problem has been solved (meaning that there

exists a computational solution). This paper contains a brief discussion and sketches a solution. A more comprehensive discussion may be found elsewhere.

The method presented will produce one single logico-semantic formula for each textual input. In case more solutions are required (and hence ambiguity is present) it is certainly possible to build together the resulting individual logic grammars.

Here we are exclusively concerned with parsing or textual analysis. Analogous considerations can be made concerning textual synthesis or generation.

We shall discuss a new method for extracting the informational content of (brief) texts formulated in natural language (NL). It makes sense to consider information extraction from NL texts to be essentially the same task as building simple kinds of information models when parsing the texts. Here we present a method that is distinguished by extreme simplicity and robustness. The simplicity makes programming of the method feasible, and so a kind of automatic program synthesis is obtained. The robustness causes wide applicability, and so the method has a high degree of generality [6,7,8,9,10,11].

2 From Dataflow to Parser Programs

It is necessary to put certain restrictions on the information flow in the attributes of a syntactic tree produced by a logic grammar, in order to consider it well-formed.

Most importantly, a consistency criterion is required: multiple instances of a rule should give rise to the same information flow locally inside the instance.

Furthermore, we require the following:

- The information flow must follow the tree structure in the sense that information may flow directly from the parent's attributes to the children's attributes or vice versa, and among the attributes of the siblings.
- The starting point of the information flow has to be a terminal word in the grammar or a vertex where a new variable is created.
- The result attribute in the distinguished vertex of the syntax tree (the root or sentential vertex) is the terminal vertex of the information flow.
- There must be a path in the information flow from each starting point to the terminal vertex.

Hence, there is no general requirement (though it may well be the case) that every attribute in every vertex should be connected to the terminal vertex of the information flow.

An input text is called exhaustive if it exhausts the grammar in the sense that the syntax tree of the text contains at least one application of each syntactic production rule in the grammar (and if it contains at least one instance of each lexical category).

When we construct a parser by means of definite clause grammars (DCGs) or other logic grammars, including the generation of a representation from a formalized logico-semantic theory, it is of course a necessary condition that the information flow in the attributes of the syntax tree corresponding to an exhaustive input text is a well-formed flow.

As an example, let us analyze the following English sentence.

"Every Swede tries to find a submarine".

Within the limits of a modestly extended first-order predicate calculus we may assign to the sentence the following three interpretations or logico-semantic representations:

$$\begin{aligned} & \exists y[\text{submarine}(y) \wedge \forall x[\text{swede}(x) \Rightarrow \text{try}(x, \text{find}(x, y))]] \\ & \forall x[\text{swede}(x) \Rightarrow \exists y[\text{submarine}(y) \wedge \text{try}(x, \text{find}(x, y))]] \\ & \forall x[\text{swede}(x) \Rightarrow \text{try}(x, \exists y[\text{submarine}(y) \wedge \text{find}(x, y)])] \end{aligned}$$

An absolutely central problem of semantics (here called the logico-semantic problem) is to assign to each input text from the appropriate linguistic universe one or several formalized semantic representations. As formalizations we shall consider here for instance logical formulae belonging to some particular logical calculus (like definite clauses or Horn clauses, first-order predicate logic, some extended first-order predicate logics, the lambda calculi, Montagovian intensional logics, situation theories, and Hans Kamp's Discourse Representation Theory) [2,3].

We shall discuss the problem of constructing a computational version of this assignment by displaying an analysis of the information flow in logic grammars. This leads to a rigorous method for the construction of a wide variety of logico-semantic assignments.

3 From Dataflow to Informational Content

We want to argue that the rigorous method described above may be implemented in a computational fashion (that is, it is fully computable). This can be done by sketching a heuristic algorithm which generates from a single exhaustive example of an input text and its corresponding intended logico-semantic representation, a logic program that translates every text from the source language into the corresponding logico-semantic representation. The heuristic algorithm should try to analyse the logico-semantic representation of the exhaustive textual input in order to build a model of the relevant information flow in the corresponding syntax tree with attributes.

Let us illustrate the method by showing how another tiny little text will be treated. The text we choose consists of four words only:

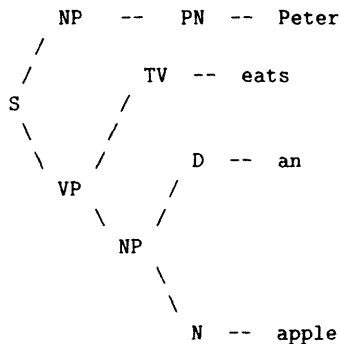
"Peter eats an apple"

Step 2 is the choice of a syntactic description. Here we select an utterly traditional context-free description like

```
S -> NP VP.
NP -> PN | D N.
VP -> TV NP.
```

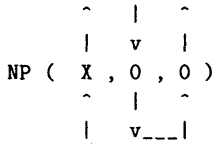
where S, NP, VP, PN, D, N, and TV designate sentence, noun phrase, verb phrase, proper name, determiner, noun, and transitive verb, respectively.

Step 3 (the analysis of information flow) is more complicated. Due to the fact that our syntax is context-free, it is possible to construct a syntactic tree for any well-formed text, so it makes sense to try to augment such a tree with further relevant information. In our little example the tree structure is



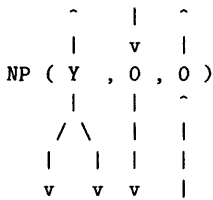
We shall illustrate the analysis by hinting at the resulting two nodes labelled NP and the one node labelled D.

The first NP node will be like this

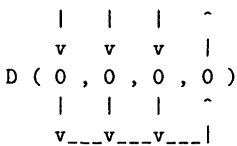


Here the node will be augmented with three arguments. The first argument is initialized to a new variable X that in turn will obtain a value from below (presumably the constant value 'peter'). The other two arguments will obtain the same value, as the value of the second argument is locally transported to the third argument as its value.

The second NP node will also get three arguments. The first argument is initialized to a new variable Y, and this (uninitialized) variable will be transported in the dataflow both upwards and downwards in two different directions (presumably to the daughter nodes, the D node and the N node). Hence we are getting something like



The D node will obtain the following local dataflow:



This means that the three first arguments will get their value from above and those values will be combined to give a value for initialization of the fourth argument.

Step 4:

From the syntax structure augmented with the dataflow, we can easily synthesize a parser program, here in the form of a definite clause grammar (DCG):

```

S( Z )    --> NP(X, Y, Z), VP(X, Y).
NP(X, Y, Z) --> PN( X ).
NP(X, Z, W) --> D(X, Y, Z, W), N(X, Y).
VP(X, W)   --> TV(X, Y, Z), NP(Y, Z, W).

```

Step 5:

It becomes an entire parser when we supply some relevant lexical information like this:

```
PN(    peter    )      --> [Peter].
TV(X, Y, eats(X,Y))    --> [eats].
D(X, Y, Z, exists(X,Y & Z)) --> [an].
N(X, apple(X))         --> [apple].
```

Step 6 (symbolic execution):

This step amounts to keeping track of each argument when evaluating and along the way change the variable names to avoid confusion (we change the name conventions so that all variables have unique names and global scopes).

```
S ( Z )
|
|-- NP ( X, Y, Z )
|   |
|   |-- PN ( X )    &  Y = Z
|   |   |
|   |   |-- Peter   &  X = peter
|   |
|   |-- VP ( X, Y )
|       |
|       |-- TV ( X, Y1, Z1 )
|       |   |
|       |   |-- eats    &  Z1 = eats(X,Y1)
|       |   |
|       |   |-- NP ( Y1, Z1, Y )
|       |       |
|       |       |-- D ( Y1, Y2, Z1, Y )
|       |       |   |
|       |       |   |-- an    &  Y = exists(Y1,Y2 & Z1)
|       |       |   |
|       |       |   |-- N ( Y1, Y2 )
|       |       |       |
|       |       |       |-- apple    &  Y2 = apple(Y1)
```

Step 7:

All the possible symbolic equations are the following:

```
Y = Z
X = peter
```

```

Z1 = eats(X,Y1)
Y  = exists(Y1,Y2 & Z1)
Y2 = apple(Y1)

```

Step 8:

This system of equations is easily solved with respect to the variable Z:

```

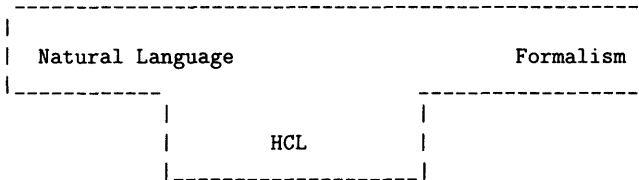
Z = Y
  = exists(Y1,Y2 & Z1)
  = exists(Y1,Y2 & eats(X,Y1))
  = exists(Y1,apple(Y1) & eats(peter,Y1))

```

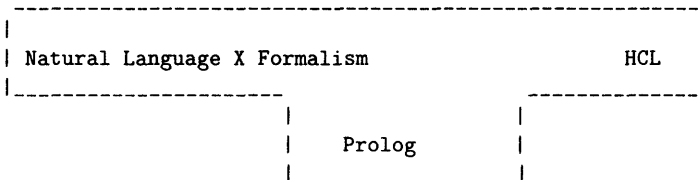
So this formula is the suggestion for the semantic representation obtained by a rigoristic and partly automated synthesis, through analysis of the information flow.

4 Relation to The Lyee Project

Our concern is the development of translating devices from a natural language description (e.g. intentions) into a formalism, shown here as an Earley diagramme:



Here the host language is Pure Prolog, or equivalently Horn Clause Logic (HCL). Our system for developing such devices may be described as a meta-parser:



Possibly it would be very advantageous to apply the Lyee method for the development of this kind of meta-parsers. And conversely, if the attempt to develop a reliable meta-parser of this kind should be successful, then such a meta-parser might possibly become an important and integral part of a possible future version of the Lyee software.

The first suggestion, applying Lyee to the development of this kind of meta-parsers, needs a little more thinking before it can be developed into a concrete project proposal, but it still may be seen as one of the major long-range goals of this project activity.

Here the focus is on the second suggestion, application of this kind of meta-parsers as an extension of the Lyee system. It should be in the shape of an experimental extension of Lyee's natural language processing capabilities.

Here we do not intend to provide any kind of assessment of Lyee's natural language processing capabilities. Instead, our purpose is to conduct some experiments with the aim of augmenting the capabilities of Lyee with respect to the logico-semantic processing of natural language utterances in the form of brief texts written in English.

When relating to the Lyee Project, we should distinguish between two cases:

Case 1: Experiments with the manual construction of logico-semantic parsers as a kind of textual preprocessing to Lyee of simple texts written in natural language (English).

Case 2: Experimental extension of Lyee with some logico-semantic functionality-constructing tools related to simple texts written in English.

Both in case 1 and in case 2, the task is the description, assimilation, and application of manual versions of the principal and central method realized by the mentioned meta-parsers. This method will be manually applied on certain carefully selected prototypical problems. Here we are talking about ongoing research to be decently described at future occasions.

The most important part of the principal and central method is an advanced kind of data flow analysis producing certain augmented syntax trees for the given input texts. These augmented syntax trees can easily and completely mechanically be transformed into executable parser programs, as discussed above.

5 Database Queries

With these methods, we investigate a variety of typical database queries. As examples we can mention

- (A) `Print every customer`

- (B) `Print the address of every customer`

- (C) `Print every customer with negative balance`

- (D) `Print the address of every supplier who supplies
any item ordered by Brooks`

Queries (A) and (C) are trivial. Others, including (B) and (D), are considerably more complex.

For each of these queries there seems to be three interesting dataflows in the unique syntax tree. One of them is immediately leading to an absurd result. It is very complicated to reproduce the underlying dataflow structures, but the reader may get an idea from the resulting representations. For instance, in example (B) it is something like

```
the(X, all(Y, customer(Y)
      => of(address(X),Y)) & print([],X))
```

Another, the most complicated flow, is awfully complex, and it leads to an almost useful representation. For example (B) it is something like

```
all(Y, the(X, customer(Y) & of(address(X),Y)) => print([],X))
```

The representation only has one major defect: it is not well-formed. Such a flaw may be repaired by a trick commonly known among computational linguists, that is so-called Quantifier Raising. And in that case we may finally obtain an acceptable representation, as a matter of fact the very representation that we would normally anticipate.

But funnily enough, there occurs also a third possibility, a dataflow intermediate between the other two. It is not so complicated as the latter, and it does not give an absurd representation like the former. In contrary it gives a representation that may be considered useful, although unexpected. Again, for example (B) it is something like

```
all(Y, customer(Y)
=> the(X, of(address(X),Y) & print([],X)))
```

Interestingly enough, the same pattern shows up with all the relatively complicated queries, including (B) and (D). It means that we may consider query (B) to be a kind of prototypical case, because it is the simplest query giving rise to the three mentioned different dataflows in the same syntax tree, with their related interpretations or resulting representations. Hence the query (B) is particularly interesting.

In addition, being in possession of the solutions for query (B) makes it very easy to reconstruct the related solutions in more complicated queries, including query (D).

Finally, acknowledgement is due to the anonymous referees for their constructive critique.

References

- [1] C. G. Brown and G. Koch, eds., *Natural Language Understanding and Logic Programming, III*, (North-Holland, Amsterdam, 1991).
- [2] K. Devlin, *Logic and Information*, Cambridge University Press, 1991.
- [3] Kamp, H. and Reyle U. *From Discourse to Logic*. Kluwer, Amsterdam, 1993.
- [4] H. Kangassalo et al., eds., *Information Modelling and Knowledge Bases VIII*, IOS, 1997.
- [5] E. Kawaguchi et al., eds., *Information Modelling and Knowledge Bases XI*, IOS, 2000.
- [6] G. Koch, A method of automated semantic parser generation with an application to language technology, 103-108, in [5].
- [7] G. Koch, A method for making computational sense of situation semantics, 308-317, A. Gelbukh, ed., *CICLing'2000, Proceedings*, Instituto Politecnico Nacional, Mexico City, 2000b.
- [8] G. Koch, Some perspectives on induction in discourse representation, 318-327, A. Gelbukh, ed., *CICLing'2000, Proceedings*, Instituto Politecnico Nacional, Mexico City, 2000c.
- [9] G. Koch, Semantic analysis of a scientific abstract using a rigoristic approach, 361-370, in [4].
- [10] G. Koch, An inductive method for automated natural language parser generation, 373-380, P. Jorrand and V. Sgurev, eds., *Artificial Intelligence: Methodology, Systems, Applications*, World Scientific, 1994a.
- [11] G. Koch, Linguistic data-flow structures, 293-308, in [1].

Using Natural Language Asymmetries in Information Processing

Anna Maria DI SCIULLO

*Université du Québec à Montréal, Département de linguistique,
Montréal, Québec, H3C 3P8, Canada*

Abstract. We posit that Information Processing is based on the recovery/parsing of functional-lexical asymmetries. Our proposal is qualitatively distinct from the standards in Information Processing, including Information Retrieval and Information Extraction, Question Answering and Text Summarization. We consider the consequences of our hypothesis for the architecture of the linguistic processor in Information Processing systems as well as for the dynamic morpho-syntactic analysis of the information supported by linguistic expressions.

Introduction

In this paper, we present a view of Information Processing based on natural language asymmetry. We consider some consequences of our proposal for Information Retrieval and Information Extraction systems, as well as for Question Answering and Text Summarization. As existing Information Processing systems are beginning to integrate articulated knowledge of the grammar of natural languages [12] [32] [35], we expect that a refinement of their architecture with asymmetry-based categorization and parsing will increase their performance.

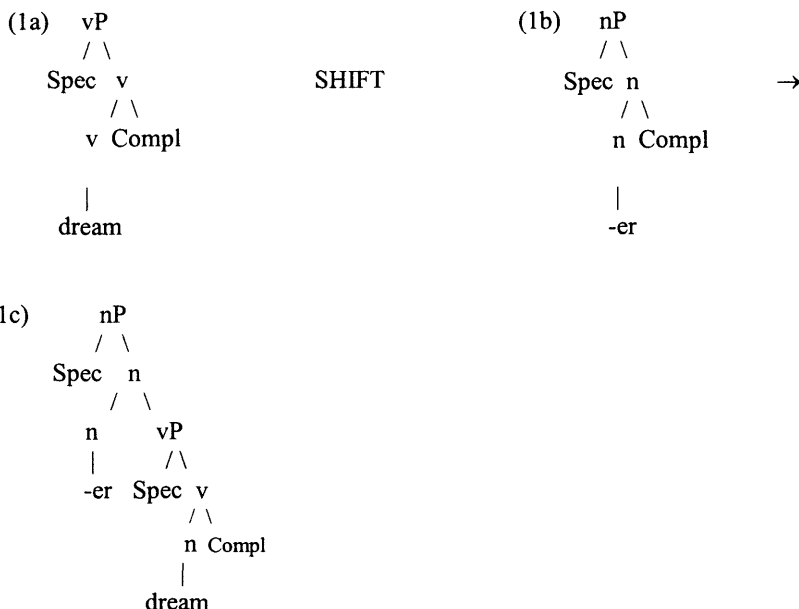
1. Natural Language Asymmetries

We take the primitives of the grammar of natural languages to be asymmetrical relations [9] [12] [14] [16] [26].¹ The sister-contain relation [9] is asymmetrical, as well as the head dependent relations [8].

According to Asymmetry Theory [16], in L(lexical)-shell the asymmetry holds between noun (N) and verb (V) heads and their dependent non-heads, the complement (object) and the specifier (subject), as implemented in (1), extending Chomsky's small vP hypothesis [9] to nominal projections L-Shells combine in the derivation of words. In (1a), the tree for the unaccusative verb *dream* is headed by a small v, and in (1b) the nominal

¹ In Set Theory, an asymmetric relation holds for the ordered pairs in a set if the pair $\langle x, y \rangle$ is part of that set but not the pair $\langle y, x \rangle$. Considering structural relations, asymmetric c-command or the sister-contain relation is asymmetric. In [9] c-command is a relation that falls out from the computational process. Thus, the operation Merge takes two elements α and β and forms a more complex one K constructed from α , β . Merge provides two relations: sisterhood, which holds for (α, β) and immediate contain, which holds for (K, α) and (K, β) . By composition of relations two new relations are derived: the relation "contain" and the relation "c-command" (sister-contain). Thus, K contains α if K immediately contains α , or K immediately contains L, which immediately contains α ; conversely α is a term of K if K contains α . And α c-commands β if α is the sister of K which contains β . See [8] for discussion.

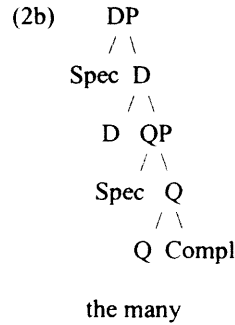
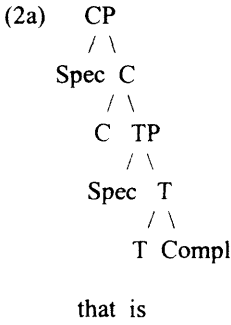
affix *-er* heads a small *n* projection. The combination of the two minimal trees by the operation SHIFT yields the morphological shell in (1c), where sister-containment holds between the nominal affix and the verbal head, before linearization onto *dreamer*. Linearization does not imply FLIPPING the structure for other languages, such as Ivie, a Niger-Congo language, where the affix precedes the root.



They are the locus of morphology specific asymmetries, such as the specifier/non-specifier asymmetry, as evidenced by the fact that subjects are excluded from compounds, ex.: *feature-checking by heads* vs. *head-checking of features*.

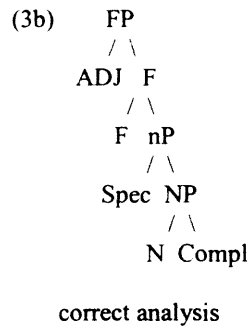
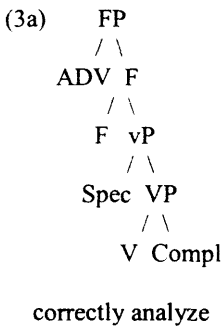
We restrict lexical categories to N and V. The other categories are functional categories. This includes Preposition (P), Adjective (A), Adverb (ADV) as well as Determiner (D), Quantifier (Q), Numeral (NUM), Demonstrative (DEM) in the functional nominal projection, and Complementizer (C), Tense (T), Modal (M) and Aspect (ASP) in the functional verbal projection.

F-shells support asymmetries amongst functional projections. This is illustrated in (2) with the relation between C and T, D and Q. In (2a) *that* is a complementizer immediately dominated by C and the tensed copula *is* is adjoined to T. In (2b), the determiner *the* is immediately dominated by D and the quantifier *many* is immediately dominated by Q.



One case of F-Shell asymmetry can be observed in the internal structure of wh-words, a point we come back to below.

Finally, F/L-Shells asymmetry restricts the relations between functional and lexical projections. F-Shells dominate L-Shells. In (3a), the ADV *correctly* is in the Specifier of FP, and the verb *analyze* has moved from V to F. Similarly in (3b) the ADJ *correct* is in the Specifier of FP and the noun *analysis* moves from N to F.



L, F and F/L Shells asymmetries partition structural descriptions in three layers: primary predication, secondary predication and quantification.

Furthermore they allow for an elegant treatment of formal parallelisms across categories. They thus provide a mean for economy in the derivations of linguistic expressions. This economy leads to greater simplicity of the overall system of rules and representations.

Shell asymmetries, overt or covert, hold for all the objects generated by the grammar, morphological and syntactic. The derivation and recovery of the asymmetries in shells are, we assume, the core of the interaction between the grammar (knowledge of language) and the performance systems (language use).

According to the Integrated Asymmetry Hypothesis proposed in [15], Universal Grammar is designed to optimally analyze linguistic expressions in terms of asymmetrical relations and Universal Parser is designed to optimally recover natural language asymmetries.

2. Natural Language Processing

2.1 Parsing

Natural language asymmetry plays a role in computational linguistics. Asymmetry was first indirectly embedded in parsers in the treatment of subadjacency in Marcus's deterministic analyzer [30] and in the treatment of c-command in Berwick and Weinberg's analyzer [5]. In principle-based parsing [4] [25] [37], there is a strict separation between the material that c-commands from the material that does not c-command the part of structure under analysis. As soon as the analyzer recognizes that the material preceding a part of structure that it integrates in the analysis does not c-command this part of the structure, it linearizes the material and directs it to interpretation: this imitates canonical analyzers, in the sense of [28].

The importance of asymmetry in computational linguistics is evident, given the central role played by asymmetrical c-command in principle-based parsers. However, the current problem in computational linguistics is to formulate a model that can process linguistic expressions efficiently and quickly. From our perspective, the use of a simplified model, based on the generation and recovery of local asymmetrical relations, constitutes the first step towards the resolution of over-generation and speed problems [17].

2.2 Information processing

The notion of asymmetry can be used successfully in Information Processing systems (retrieval and extraction).

In this area, it is urgent to find efficient solutions to the problems of users who consult vast data banks such as the Internet to obtain accurate information on specific subjects. Although it is no longer currently necessary to be familiar with the use of operators or logical connectors to query Internet, data search results are often inaccurate, too numerous or too vague because they are solely based on logic and vicinity relations between searched words or sometimes even between character chains composing these words. All other relations between these words are usually omitted, including semantic, syntactic and morpho-syntactic relations. Therefore, even the most reputable Internet search systems are limited to words, meaning that no preliminary linguistic processing is executed. When other services execute linguistic pre-processing, the index is generally made-up by nothing more than the lemmatization of the vocabulary. In most cases, the so-called 'natural language' search is based on statistical techniques without any systematic appeal to natural language processing techniques. It is only recently and on a low scale that certain data search systems have begun to integrate and use morpho-syntactic knowledge in search engines.

The current problem in Information Processing is to improve precision. Natural language processing methods offer a new solution to this problem, since it is now known that retrieval based solely on statistical techniques has reached its limit. It has also notably been shown that the use of a stemming algorithm during data search indexing and query analysis provides a part of the solution to the search tool efficiency problem [33]. Another part of the solution to the current problem is the indexing of nominal expressions, which helps to delimit the meaning (referent) of the data to be retrieved [36]. These are reasons to believe that the integration of asymmetrical relations into search tools increases their precision.

Information Processing tools whose methodology is based on natural language processing techniques are more accurate and easier to use than those based exclusively on

Boolean operators or other such functions based on the recognition and retrieval of chains of character. The formulation of queries in natural language and the use of morphological and syntactic relations between the words contribute to the improvement of the performance and usability of search engines.

3. Natural Language Asymmetry-Based Information Processing

The approach that we take with respect to Information Retrieval and Extraction is based on natural language processing. According to this approach, retrieval and extraction call upon natural language morpho-syntactic and syntactic knowledge.

Our approach contrasts with the Boolean approach which is commonly used in Information Processing. In Information Retrieval systems for example, the indexing of the query consists in its translation into Boolean proper forms, while the indexing of the documents consists in the identification of document profiles by means of the selection of descriptors (significant words) for document profiles using a descriptor dictionary.

It is possible to optimize Information Processing engines by applying natural language processing analyses to queries, particularly morpho-syntactic analysis, and terms expansions, thus providing the means to construct nominal structures and reduce ambiguity.

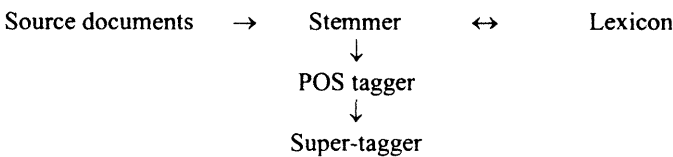
In the following paragraphs we consider the consequences of the integration of asymmetry-based natural language properties in Information Processing systems.

The role of functional categories (the so-called “stop words” retrieved from processing in standard Information Retrieval systems) is crucial in our view.

3.1 The architecture of linguistic processor

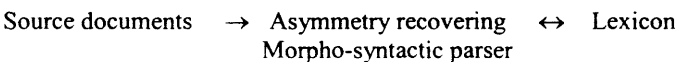
The architecture of the linguistic processor in Information Processing systems generally includes Stemmers, Part of speech (POS) taggers as well as Super-taggers, as schematized below.

(4) Linear linguistic processor



A first consequence of our proposal is that the architecture of the linguistic processor may depart from the standard linear ordering of modules in (4) and turn to dynamic linguistic processing in (5), where the morphological analysis is performed interactively in conjunction with the processing of the morpho-syntactic asymmetries.

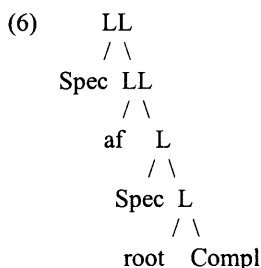
(5) Dynamic linguistic processor



Asymmetry-based parsing increases the accuracy of Information Processing. Stemming algorithms are generally ad hoc and offer mediocre results (Porter algorithm

[34]) and to some extent KIMMO [24] [2]. If stemming is performed on the basis of the recovery of morphological asymmetries, including the identification of the categorial head of words, standardization is improved.

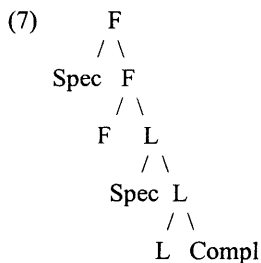
The recovery of the L-shell asymmetries, as depicted in (6), is crucial in the determination of the standard form of words, as a given lexical affix may combine with roots on the basis of their configurational properties [14] [16]. The recovery of the asymmetrical categorial structure of derived words improves stemming, in interaction with sub-word part of speech (POS).



There is evidence to the effect that words are configurations [15] [22]. Asymmetry plays a role in the structuring of derived as well as non derived words. Both derivational and inflectional affixes head words, given the Relativized Head Rule [18] according to which the category of the rightmost element specified for a categorial feature is the category of the word it is a part of, ex.: *process_V-ing_N*, *process_V-or_N-s_{Npl}*.

The F/L Shell asymmetry is also crucial in the so-called POS tagging. Most POS tagging modules are statistically based, or based on distributional repair strategies [6]. In our view, POS tagging is a relational process and not a categorial one. It dynamically interacts with stemming, as inflectional and derivational heads provide categorial information for POS tagging. In our view, POS tagging is based on the configurational morpho-syntactic asymmetry relating a functional head to its non-head. Thus, it is the asymmetrical relation between a complementizer and the categories contained in its complement domain or the asymmetrical relation between a determiner and the categories contained in its complement domain that are determinant, ex.: *to_V cut an apple* vs. *the_A cut apple*; *it_V contained an antecedent* vs. *antecedent_A contained deletion*.

Asymmetry is also at play in the identification of the head of syntactic constituents (Super-tagging). This process is not independent from POS tagging and stemming, as a syntactic constituent must have a head. Assuming that constituents are binary branching projections, a syntactic head is asymmetrically related to its non-head. The recovery of F/L-Shell improves syntactic analysis. Misanalysis may arise for example from N/V conversion in English, ex: *how do you monitor costs?*, where taken in isolation both *monitor* and *costs* can be analyzed as parts of N or V projections. Optimal processing can be achieved with F/L-Shells.

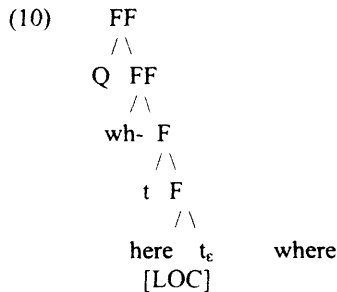
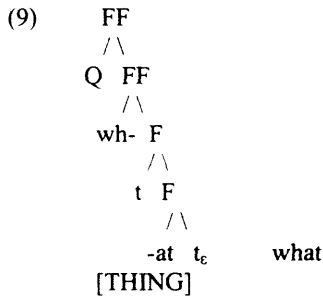
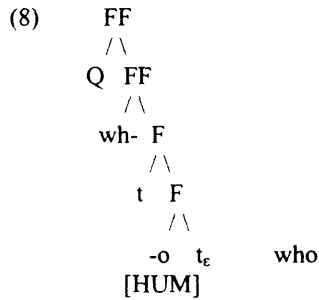


The dynamic recovery/parsing of Shell asymmetries allows for a reorganization of linearly organized Information Processing systems including independent stemming, POS tagging and Super-tagging modules, as these modules present three sides of the same asymmetry recovering process.

3.2 Question answering

Asymmetry recovery/parsing also allows to optimize open-question answering systems.

We assume that *wh*-words include an asymmetrical FF-Shell, as proposed in [14]. In these shells, a *wh*-variable is the head of the upper layer of the Shell. The restrictor of the variable includes interpretable features such as human ([HUM]), thing ([THING]), and location ([LOC]), morphologically spelled out by the morphemes *-o*, *-at*, *here*, sits in the lower layer of the shell. See (8)-(10) below for *who*, *what* and *where*. In these Operator-Shells, the question operator (Q) is in the Specifier of the upper layer of the Shell (FF), and the truth value types of denotation *t* in the lower layer express the fact that a question operator relates a proposition (*t*) to a set of possible answers (*t_e*) to the question.



The structured morpho-syntactic F-Shells allow for an optimal search and retrieval of the relevant answer, which in interaction with the syntactic analysis of the documents will extract relevant [HUM], [THING] or [LOC] ex.: *Mary danced the waltz in Vienna. Who danced? Mary.; What did Mary dance? The waltz.; Where did Mary dance the waltz? In Vienna.*

Each member of the wh paradigm differs with respect to the feature structure in the lower stratum of the FF-shell. The recovery of the morpho-syntactic functional asymmetry of wh-words ensures the optimization of open-question answering systems.

3.3 Summarization

Another consequence of asymmetry-based IP is that it also leads to enhance the precision of text summarization.

In this case the F/L-Shell asymmetry is crucial and its recovery provides a way to distinguish what we will call the 'Topical' vs. 'Non-Topical' information. This can be done by identifying a set of asymmetrical relations that support topical information and a set of asymmetrical relations that support non-topical information. Some of these relations are stated below.

- (11) **Topical asymmetries:**
primary predication,
restricted modification
- (12) **Non-topical asymmetries:**
unrestricted modification
pronominal anaphora
quantifier-binding
- (13)
$$\begin{array}{rcl}
 & \text{FF} & \text{Non-topical} \\
 & /\backslash & \\
 & \text{FF} & \\
 & /\backslash & \\
 \text{FF} & \text{F} & \text{Topical} \\
 & /\backslash & \\
 & \text{F} & \\
 & /\backslash & \\
 & \text{F} & \text{L} \\
 & /\backslash & \\
 & & \text{L} \\
 & /\backslash & \\
 & & \text{L}
 \end{array}$$

The identification of Topical asymmetries in linguistic expressions (queries, texts, collection) enhances the precision of summarization, which in turn may constitute the basis to enhance the precision and the recall of Information Processing systems.

If Text Summarization is performed on the basis of the parsing of Topical asymmetries, it is possible to identify the relevant information with respect to a template (Information Extraction) as well as the set of addresses (Information Retrieval) that are most relevant with respect to a query.

Our approach contrasts with the *statistical* [31] and *deep understanding* [1] approaches. The former extracts the salient information using word frequency and

positional cues, while the latter uses deep understanding in restricted domains. Our proposal also subsumes standard editing parameters for summary construction, such as the (titles, sub-titles, etc.) and develops a natural language processing approach to summarization.

4. Conclusion

It is generally the case that Information Processing reduces to the processing of the actual words that are part of linguistic expressions. In our view the information is supported by abstract shells. The consequences of our proposal range over the properties of the architecture of Information Processing systems and the properties of its different applications.

Our proposal provides Information Processing systems with the necessary theoretical and linguistic tools to improve their performance. The integration of asymmetry-based morpho-syntactic analysis during indexing and search increases the quality of extraction and retrieval systems. The uniformity of the configurations and the rule schemata reduce the search space and promote the rapidity of computational treatment.

5. Acknowledgements

This work is supported in part by funding from the Social Sciences and Humanities Research Council of Canada to the Asymmetry Project, grant number 214-97-0016, as well as by Valorisation-Recherche Québec, grant number 2200-006.

References

- [1] R. Alterman, "Text Summarization", *Encyclopedia of Artificial Intelligence*, Shapiro, S. (editor), John Wiley & Sons, Vol. 2, 1992, pp. 1579-1587.
- [2] E. Antworth, PC-KIMMO: A Two Level Processor for Morphological Analysis, Dallas, TX: Summer Institute of Linguistics, 1990.
- [3] A.T. Arampatzis, T. Tsois and C.H.A. Koster, IRENA: Information Retrieval Engine Based on Natural Language Analysis, RIAO 97 Proceedings, McGill University, 1997.
- [4] R. Berwick, "Principles of Principle-Based Parsing", *Principle-Based Parsing*, R. Berwick, S. Abney and C. Tenny (eds.), Dordrecht: Kluwer, 1991.
- [5] R. Berwick and A. Weinberg, *The Grammatical Basis of Linguistic Performance*, Cambridge Mass.: The MIT Press, 1986.
- [6] E. Brill, "Some Advances in Transformation-Based Part of Speech Tagging", *Proceedings of the 12th National Conference on Artificial Intelligence, AAAI 94*, 1994.
- [7] E. Brill, "A Simple Rule-Based Part of Speech Tagger", *Proceedings of the Third Conference on Applied Natural Language Processing, ACL*, 1992.
- [8] M. Brody, "Mirror Theory", *Linguistic Analysis* 31:1, 2000.
- [9] N. Chomsky, *Minimalist Inquiries*, Ms. MIT, 1998.
- [10] N. Chomsky, *The Minimalist Program*, Cambridge, Mass.: The MIT Press, 1995.
- [11] C. Collins, *Local Economy*, Cambridge, Mass.: The MIT Press, 1997.
- [12] A. Copestake and T. Briscoe, "Semi-Productive Polysemy and Sense Extension", *Lexical Semantics. The Problem of Polysemy*, Pustejovsky & B. Boguraev (eds.), Oxford University Press, 1996.
- [13] S. Deroose, "Grammatical Category Disambiguating by Statistical Optimization", *Computational Linguistics* 14, 1988.
- [14] A.M. Di Sciullo, *Asymmetry in Morphology*, Cambridge, MA: MIT Press. Forthcoming.
- [15] A. M. Di Sciullo, "Formal Context and Morphological Analysis", *CONTEXT 99*, P. Bouquet & al. (eds.), Springer Publishers, 1999b, pp. 105-119.
- [16] A.M. Di Sciullo, "The Local Asymmetry Connection", *MIT Papers on Linguistics*, Cambridge, Mass: The MIT Press., 1999a.

- [17] A.M. Di Sciullo and S. Fong, *Morpho-Syntax Parsing*, Ms. UQAM and NEC Princeton, 2000.
- [18] A.M. Di Sciullo and E. Williams, *On the Definition of Word*, Cambridge, Mass.: The MIT Press, 1987.
- [19] M. Diesing, *Indefinites*, Cambridge, Mass.: The MIT Press, 1992.
- [20] S. Fong, *The Computational Implementation of Principled-Based Parsers*, *Principle-Based Parsing*, R. Berwick, S. Abney & C. Tenny (eds.), Dordrecht: Kluwer, 1991.
- [21] R. Gaizauskas and A. Roberston, "Coupling Information Retrieval and Information Extraction: A New Text Technology for Gathering Information from the Web", *RIAO 97*, Montréal, 1997, pp. 356-370.
- [22] K. Hale and J. Keyser, "On the Restricted Nature of Argument Structure", *MIT Papers on Linguistics*, Cambridge, Mass: MIT press, 1999
- [23] M. Halle and A. Marantz, "Distributed Morphology and the Pieces of Inflection", *The View from Building 20*, Cambridge, K. Hale and J. Keyser (eds), MA: MIT Press, 1993.
- [24] L. Karttunen, "KIMMO: A General Morphological Processor", *Texas Linguistic Forum* 22, 1983.
- [25] M. Kashket, "Parsing Walpiri. A Free Word Order Language", *Principle-Based Parsing*, R. Berwick, S. Abney and C. Tenny (eds.), Dordrecht: Kluwer, 1991.
- [26] R. Kayne, *The Antisymmetry of Syntax*, Cambridge, Mass.: The MIT Press, 1994.
- [27] H. Kitahara, *Elementary Operations and Optimal Derivations*, Cambridge, Mass.: The MIT Press, 1997.
- [28] D. Knuth, *On the Translation of Languages from Left to Right*. *Information and Control*, pp. 607-639.
- [29] C.G. Marken, "Parsing the LOB Corpus", *Association of Computational Linguistics Annual Meeting*, 1990.
- [30] M. Marcus, *A Theory of Syntactic Recognition for Natural Language*, Cambridge, Mass.: The MIT Press, 1980
- [31] M. Maybury, "Generating Summaries from Event Data", *Information Processing & Management*, 31, (5), 1995, pp. 735- 751.
- [32] M.T. Pazienza, (ed) *Information Extraction. A Multidisciplinary Approach to an Emerging Information Technology*, Springer Publishers, 1997.
- [33] R. Pohlmann and W. Kraaij, "The Effect of Syntactic Phrase Indexing on Retrieval Performance for Dutch Texts", *RIAO 97 Proceedings*, McGill University, 1997.
- [34] M.F. Porter, *An Algorithm for Suffix Stripping Program*, 14.3, 1980.
- [35] J. Pustejovsky, B. Boguraev, M. Verhagen, P. Buitelaar, M. Johnston, "Semantic Indexing and Typed Hyperlinking", *Natural Language Processing for the World Wide Web. Papers from the 1977 AAAI Spring Symposium*, AAAI Press, 1997.
- [36] A. Schiller, "Multilingual Finite-State Noun Phrase Extraction", *Proceedings of the ECAI96 Workshop*, 1996.
- [37] P. Stabler, "Avoid the Pedestrian's Paradox", *Principle Based Parsing: Computation and Psycholinguistics*, R. Berwick, S. Abney and C. Tenny (eds.), 1991, pp. 199-239.

This page intentionally left blank

Chapter 3

Lyee-Oriented Software Engineering and Applications

This page intentionally left blank

Engineering Characteristics of Lyee Program

Ryosuke HOTAKA

14-21 Fukasawa 1-chome, Setagaya Tokyo 158-0081 Japan

Abstract. An alternative program generation experiment employing Lyee principle is explained. Whereas Lyee system employs autonomous or kind of harmonised distributed approach, we adopt central or top down approach. The merit of top-down approach is its natural control mechanism that is popular among current engineers though it sacrifices deeper philosophical thinking of Lyee. A very naive example is used to show that there exist invariant engineering characteristics which enable program generation of any kind. Future directions of open source activities are mentioned.

Keywords: automatic program generation, sequence invariance, side effects caused by a change of a word, universal structure of logic accompanying a word, stability, parametric execution of the kernel of an application program, serialized visualization of control logic

1. Introduction

Since development of software requires a lot of efforts and cost, a program generator is a dream of many users. Historically, several software packages with names that remind us program generator appeared. RPG of AS400 is one of such examples. Their common techniques are parametric generation of programs. E.g., they extracted common patterns of making reporting programs and majority of report generation can be produced by careful selection of parameters. But this approach fails if unexpected processing was required. The author remembers that he always use exceptional own coding facility of RPGII of System/38, the predecessor of AS400, and not the parametric specification.

Until Lyee appears, no software packages could be categorised as general-purpose program generator. Since a general-purpose program generator requires a sufficiently large coverage of application field, naive approach of common patterns with parametric selection will not work. Some drastic ideas are necessary for this. To the author's eyes, Lyee is the first to bring such ideas.

Lyee depends on some philosophical theses. Because of this philosophical nature, the resulting software developing tools are deeply influenced by the original hypothetical concept. It is good in that it has much flexibility in applying Lyee in practical application program development, but at the same time it introduced a very hard hurdle for current engineers to overcome.

The first objective of this paper is to show that a similar generator named OpenLyee (software generating system) can be constructed using traditional engineering technologies. As the name indicates, it depends on deep insight that Lyee first revealed.

It particularly followed the ideas presented in [1] and [2] and presented them in a different and hopefully easier-for-traditional-engineer fashion. Especially, we newly devised automatic control of base structure execution, thus automating users' system design works, e.g., it is not necessary to define process route diagram ([2]). The author must admit,

however, that OpenLyee is in its experimental state and needs a lot of improvements in future.

Login	<input type="button" value="Submit"/>
Password	<input type="text"/>
\$message	<input type="text"/>

Deduction	<input type="button" value="Confirm"/>
AccountNo	<input type="text"/>
Deduction	<input type="text"/>
Balance	<input type="text"/>
\$message	<input type="text"/>

Fig. 1 Two gui screens

OpenLyee concepts differs slightly from those of Lyee's because it employs a different kernel logic to control an application program. Therefore, the author restated the concept again if it is necessary.

In section 2, an example application is explained. The implementation of this application is sketched in section 3. The source code of this application program would have been generated by OpenLyee. During preparation of this paper, some experimental coding was tried to ascertain the feasibility of our approach. It is observed that there are several common control structures that are independent from applications. Owing to this characteristic, it becomes possible to develop a general-purpose application program generator. This invariance is remarkable engineering characteristic that come from Lyee principles and are stated in section 4. Currently OpenLyee presupposes several conditions to simplify the development environment in section 5. Our development environment and the current state and the future direction of development are briefly mentioned in section 6. Finally, conclusions and further studies are stated in section 7.

2. Explanation of the Example Application Program

2.1 User Interaction of the Example

Consider the following application. There are two gui (graphic user interface) screens: Login and Deduction (Fig.1).

We consider a case where a customer of a bank is supposed to confirm if he can withdraw certain amount of money from his account. First, the Deduction screen which corresponds to the base structure Deduction is displayed. Then the customer enters his account number in the AccountNo field and the amount of money he wishes to withdraw in Deduction field. When the Confirm button is clicked, the Login screen is displayed and the Deduction screen is freed. The user is prompted to enter his password in Password field. When the Submit button is clicked, data in the Login screen is sent to the application and the balance after the withdrawal of amount of money entered in Deduction field will be shown at Balance field in the Deduction screen. If the current balance is not sufficient to withdraw the amount of money entered in Deduction field of Deduction screen, 0 will be displayed at Balance field and the message "insufficient fund" appears at the \$message field.

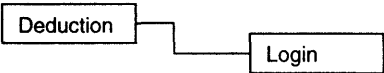


Fig. 2 Process Route Diagram of the application

Login		
W04	W02	W03

Deduction		
W04	W02	W03

Fig. 3 Pallets in a Base Structure

2.2 Lyee Program Structures

In the example, the whole processing logic is divided into two Base Structures: one corresponding to Login screen and the other corresponding to Deduction screen. We name them following the screen names, thus getting Login and Deduction.

If we follow the convention of Lyee, these two base structures are visually connected as in Fig. 2. It is called a process route diagram ([2]). (Note. In fact, the first base structure that gets control is Deduction. This may seem abnormal, but this is a logical consequence because base structure execution is automatically controlled. See section 2.5 for its reason.)

Usually a process route diagram is designed by users, but in OpenLyee it is determined automatically.

Each base structure is a unit or module of processing. It is further decomposed into 3 processing units (Fig. 3). The 1st one is called W04 pallet, the 2nd W02 pallet and the 3rd W03 pallet.

Roughly speaking, W04 is responsible to produce output, W02 to accept input and W03 remaining jobs.

2.3 How an application program works

The minimum data unit in Lyee is called a word and each word has its corresponding programs. A word corresponds to a variable in the application system. Informally, Lyee could be described as follows: An application program conforming to Lyee starts with an initial state where every content of word is empty. It repeats pallets and base structure processing systematically to raise the count of resolved words until there remain no other ways to raise the count, then it stops. Of course, if all the words are resolved and have non-empty values, that is the end of processing, but it may not reach that state and stops with some words or all words without values if conditions to find non-empty values are not met. Lyee called this state synchronicity. It is a kind of stable state. Informally, Lyee is a method that guides every word from its initial state to its stable state systematically. Detailed explanation is given in section 3.

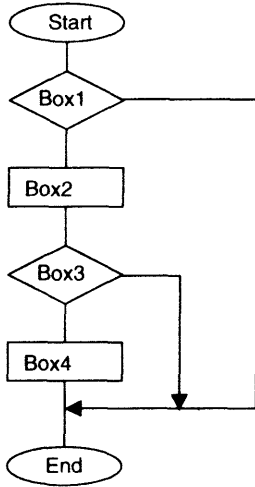


Fig. 4-1 Logic of W04- or W02-coordinate

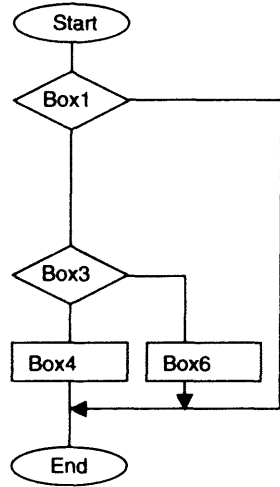


Fig. 4-2 Logic of W03-coordinate

2.4 Words

Each word belongs to one and only one base structure. Since a base structure is decomposed into 3 pallets, each program belonging to the base structure is decomposed into 3 parts and the 1st one belongs to W04 pallet, the 2nd one belongs to W02 pallet and the 3rd one belongs to W03 pallet respectively.

E.g., Balance has 3 parts of program. The 1st part is assigned to W04 pallet with the program logic of Fig. 4-1, the 2nd part is assigned to W02 with the program logic of Fig. 4-1 and the 3rd part is assigned to W03 with the program logic of Fig. 4-2. To distinguish a word itself from one of its part, we call the latter W04-coordinate, W02-coordinate or W03-coordinate of the word respectively.

If the name of a word is Password, for example, the W04-coordinate of the word is named Login.W04.Password. Thus in our example, 12 coordinates appear, i.e., Login.W04.Password, Login.W02.Password, Login.W03.Password, Deduction.W04.AccountNo, Deduction.W02.AccountNo, Deduction.W03.AccountNo, Deduction.W04.Deduction, Deduction.W02.Deduction, Deduction.W03.Deduction, Deduction.W04.Balance, Deduction.W02.Balance, and Deduction.W03.Balance.

\$message in both Login screen and Deduction screen looks like a word, but it is not in exact sense. It is a special variable to hold error messages. Its details are omitted in this paper. Submit in Login screen and Confirm in Deduction screen are buttons to notify completion of inputs and are not words in the above sense.

The logic of each coordinate of a word is exhibited in Fig. 4. Fig. 4-1 logic is used in both W04-coordinate and W02-coordinate. Fig. 4-2 logic is used in W03-coordinate. They are different from the current Lyee method, but we use simpler (and in fact older) version. E.g., our Fig.4-1 lacks Boxes 5, 6 and 7 (see e.g. Fig.10 in [1]). The main difference is in the way pallets and base structures are controlled. We just raise the count of non-empty words in Box4. This has already been discussed in section 2.3 and will be discussed again when the stability of states of an application matters (see section 3.4).

In Box1 of Fig. 4-1, if the value of the word is empty (i.e., worth calculation) and it is ready to calculate its value then go to Box2 else go to End.

In Box2 of Fig. 4-1, temporary value of the word is calculated.

In Box3 of Fig. 4-1, if the calculated result in Box2 is not acceptable, go to End else go to Box4.

In Box4 of Fig. 4-1, the count of non-empty words is raised by 1.

In Box1 of Fig. 4-2, if the W03-coordinate is non-empty or if other base structures (to calculate the corresponding value of the word) have been scheduled then the control goes to End.

In Box3 of Fig. 4-2, if every word appearing in the derivation rule (see 2.5) of the corresponding word has non-empty value go to Box4 else go to Box6.

In Box4 of Fig. 4-2, change the state of the word ready to derive its value and raise the count of the number of non-empty words by 1.

In Box6 of Fig. 4-2, the flag showing the existence of a scheduled is set.

2.5 Derived Word and Deriving Word

In OpenLyee, a word is a derived word if the value is not given at the beginning but only the derivation rule is given. A derived word W needs some values of other words, say W1, W2, ..., Wn. They are called deriving words of W. One of Wi may itself be a derived word and it needs deriving words again. Derivation rules are specified in the definition of W04-coordinate. Suppose the set of words be nodes of a graph and directed arc from Wi to W be defined.

If a base structure has no derived words, the corresponding application program will not go searching its deriving words and the process will not continue further. Since a non-empty value of Deduction.W04.Balance is wanted, the base structure Deduction is selected first in our example application.

Main jobs of users of OpenLyee are to give a derivation rule for each W04-coordinate of a derived word. pallet. In our example, they are as follows:

Calculation rule of each calculated word in Deduction

for	Deduction.W04.AccountNo,	Deduction.W02.AccountNo
for	Deduction.W04.Deduction,	Deduction.W02.Deduction
for	Deduction.W04.Balance,	max("0", ColVal(Login.W02.Password, "Account", "Balance")-Deduction.W02.Deduction)

(Note. ColVal is a user-defined database access function which returns current balance of the account specified by Deduction.W04.AccountNo provided the password supplied by the user matches the password in the database. We will not go into detail here.)

Calculation rule of each calculated word in Login

for	Login.W04.Password,	*****
-----	---------------------	-------

The method that uses relationships between calculated data and data used for calculation was also tried in [6] to design a database.

Users must define or prepare other constructs, such as forms used in gui screen or I/O interface for database access function, but details are out of scope of this paper.

BSStack	
Base Structure	Level
Deduction	0

Fig.5-1 The content of BSStack when Deduction is entered

BSStack	
Base Structure	Level
Deduction	0
Login	1

Fig.5-2 The content of BSStack when Login is added

BSStack		
BS	Level	BSprvNoOfTrueWrds

Fig. 6 BSStack

2.6 Control of Base Structures

Multiple base structures are used in an application program and they are controlled using BS stack. Suppose base structure Deduction gets control first, then the BSStack is shown as (Fig. 5-1). The level of Deduction is 0 because it is the first base structure to get control. Further, suppose W03 of Deduction finds that base structure Login is necessary (in our case, needs to know the password), then the BSStack changes into Fig. 5-2. The level of Login is 1 because it is called by level 0 base structure.

If Deduction needs other base structures, they would have entered after Deduction with Level=1. BSStack is used to control base structure execution so that necessary base structures are called and unnecessary calling of base structure are suppressed.

3. Detailed Explanation of a Simple Example

Appendix: Step-by-step investigation of the example constitutes the core of our paper. We complement explanation that was not mentioned in the appendix.

3.1 BSStack

Another field or column BSprvNoOfTrueWrds (Base Structure’s Previous Number of True Words) is added to the BSStack. Now the BSStack looks like Fig. 6:

BSprvNoOfTrueWrds is used to remember how many true words existed when a new base structure is created. The meaning will become clearer in *section 3.3*.

The BSStack of the example seems too simple to be useful. As a more practical case, consider the next process route diagram (Fig. 7).

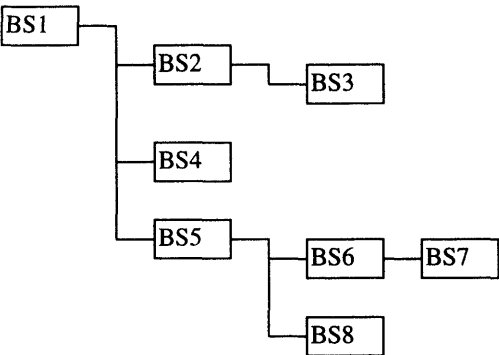


Fig. 7 A Process Route Diagram (more complex case)

BSStack	
Base Structure	Level
BS1	0
BS2	1
BS4	1
BS5	1

Fig. 8-1
Contents of BSStack

BSStack	
Base Structure	Level
BS1	0
BS2	1
BS4	1
BS5	1
BS6	2
BS8	2

Fig. 8-2
Contents of BSStack

BSStack	
Base Structure	Level
BS1	0
BS2	1
BS4	1
BS5	1
BS6	2
BS7	3

Fig. 8-3
Contents of BSStack

OpenLyee does not need the definition of a process route diagram by users because it is determined automatically. Therefore, Fig. 7 shows the resultant process route diagram created internally during the execution of an application program.

Suppose 3 base structures BS2, BS4 and BS5 are scheduled at the end of BS1. The contents of BSStack is Fig. 8-1 (where column BSprvNoOfTrueWrds is omitted).

Suppose further BS5 is executed and 2 base structures BS6 and BS8 are scheduled at the end of BS5. Then the contents of BSStack is Fig. 8-2. After execution of BS8, if BS7 is scheduled, then the contents of BSStack is Fig. 8-3. Thus BSStack always represents a tree.

3.2 W02 pallet

Different from ordinary application program of Lyee, our application program has only 1 W02 pallet executed per 1 execution of base structure (see Fig. 10 of Appendix). This is the assumption of this paper.

BSStack		
BS	Level	BSprvNoOfTrueWrds
Deduction	0	6
Login	1	

Fig. 9 The Content of BSStack at *11

3.3 Serialized view of Base Structure Execution

Each of Lyee’s constructs e.g., base structure, pallet, word has multiple roles in execution time. At a particular point of time, however, only a few roles are relevant whereas other roles are waiting to become relevant at another point of time. This dynamic relevance of roles makes the logic of Lyee difficult to grasp as a whole. This difficulty can be overcome when we serialize the behaviour of Lyee and follow the logic as time goes. The visualization of logic in time horizon is stated in Fig. 10 of Appendix and is a crucial part of the paper.

In Fig. 10 of Appendix, base structure executions are serialized in the order of execution. In this way, we visualized and distinguished the logic as time goes. Lyee’s program and a human cell resemble one another. It is equipped with multiples of functions, and a function become evident when it confronts a particular stimulus. Whereas when it is investigated in inactive state, it is not obvious to understand its functions. Serialized view of base structure execution alleviates this difficulty. We have attached a comment number (e.g. *1, *2,...) to a point where explanation is attached.

3.4 Stability Check

One of the significant characteristic of Lyee program is the capability of testing whether an application program is stable or not. There would be many methods of implementation of stability check, but we adopted the checking the saturation of monotone increasing number of true words (as was stated in Section 2.3 before). The checking is performed after every base structure execution (e.g. *5, *8, *11 in Fig. 10).

We have introduced a new check mechanism for stability. The general logic is as follows:

- (1) Initialise the count of true words
- (2) Initialise the temporary count of true words equal to the count of true words
- (3) Begin processing a group of codes (during processing raise the temporary count of true words if a new word changes its value from empty to non-empty)
- (4) At the end of the processing, the generated application program checks if the temporary count of true words is equal to the count of true words. If it is unequal that means the group of codes is not stable and repeat (3).

Stability check appears two places in the application program: the point after execution of words in W04 and the point after the execution of a base structure. Stability checks make the application program independent from the sequence of executions of words and dispense with application dependent control mechanism.

At *5, W02 has changed because words `AccountNo` and `Deduction` have been entered. At *8, W04 has changed because the above words are copied to corresponding words in W04. In both cases, the base structure `Deduction` has changed, therefore it is not stable yet. Whereas at *11, no words have changed, therefore it is stable. We could apply similar reasoning to remaining points: *16, *19, *22 and *25.

At *11, `Deduction` is found to be stable, but since there remains the word `Balance` empty, the application program will continue to check if it can make the value of `Balance` non-empty. `Balance` is derived from two words: `Login.W02.Password` and `Deduction.W02.Deduction`. `Deduction.W02.Deduction` has already non-empty values, but `Login.W02.Password` still has empty-value. Therefore, base structure `Login` is stacked in `BSSStack` with `Level = 0`. In order to check the stability of `Deduction` later (*22), `noOfTrueWord` is stored in `BSprvNoOfTrueWrds`. After this preparation, control is transfer to `Login`.

The contents of `BSSStack` at this point is shown as Fig. 9.

At *22, `Login` is found to be stable and there are no words that remain empty, therefore control returns to `Deduction` again with `noOfTrueWord=9` and after that the value of `BSprvNoOfTrueWrds` is recovered. Since the value of `noOfTrueWord` is different from (in fact greater than) the recovered value of `BSprvNoOfTrueWrds` (=6), a change has occurred in the execution of `Login`, and `Deduction` might not be stable yet. Thus `Deduction` is re-executed again.

At *28, `Deduction` is stable and there are nothing left to be examined, therefore the application program stops.

4. Engineering Characteristics of Lyee Program

Though Section 3 is one particular example, regular logics appear repeatedly in a disciplined way. This characterizes Lyee Program and this regularity assures the automatic program generator. Here, we note several of them.

4.1 Parametrical execution of the core of generated application program

We state all the logics in Section 3 in such a way that `OpenLyee` (the application program generator) can specify various objects e.g. words, pallets and base structures parametrically. That means that `OpenLyee` generates always the same template code for base structure control. The remaining task for it would be the setting of environmental conditions.

4.2 Sequence invariance and automatic base structure control

As a consequence of section 3.4 a generated program is independent from sequence of words execution and users need not worry about the sequence of base structures execution.

4.3 Influence of addition and deletion of words

Users' specifications of an application program are restricted to

- (1) Definitions of data format and definitions of calculation algorithms of words,
 - (2) Definitions of I/O interfaces (screen forms of gui and interface area of file I/O),
- and
- (3) Definition of ad hoc functions used in e.g. scientific calculations or database manipulations.

The application program will then be automatically generated.

Therefore, debugging of an application program reduces to identifying an inadequate definition. Other considerations such as the modification of program control logic are not required.

4.4 Stability check

Lyee program succeeded in controlling application program even if users do not design how to do it. The fundamental principle is to loop word programs indefinitely until needed words are filled with non-empty values. Therefore, there must be some mechanism that stops unnecessary loops. Lyee introduced stability check for this purpose and it is a characteristic of Lyee-like automatic program generation.

5. Assumptions

We deliberately confined ourselves to simple cases. E.g., we restricted variations of application programs when there are some means to realise the same functionality since our major objective is to experiment the feasibility of OpenLyee.

5.1 Preparation of gui's and databases at the beginning

All the gui's and databases are prepared at the beginning. No capabilities of dynamic creation of gui's and dynamic creation of database usages are considered during the execution of an application program.

5.2 Correspondence of gui and base structure

We restricted more than one gui for a base structure.

5.3 At most one read operation in an execution of a base structure

Under the assumption 5.2 we wonder if more than one read is necessary for each execution of a base structure. Therefore we set this assumption to see if our assumption is appropriate or not.

5.4 Database capability

Output to databases was not considered.

5.5 Ad hoc functions

Any other ad hoc functions such as special user-defined functions of buttons are future subjects.

5.6 Sharing of a base structure

We assume there is no base structure such that it appears twice in the tree of the BSSStack.

6. State of Development

We began our partial implementation using Java language. Many program logics are forced to change to adapt to Java language to author's frustration, but after some acquaintance of Java, the program logic seems to become more systematic and readable.

Material presented in the appendix comes from the result of partial coding of the example. During the design of the example, we take a lot of precaution so that the example can be naturally generalized to the program generator.

7. Conclusion and Further Studies

The major objective of this paper is to understand Lyee itself. To the author, 2 years of struggle of learning is not sufficient to understand every detail of Lyee. The author, therefore came to a conclusion that the core concept of Lyee would best be learnt by developing Lyee-like program generator himself.

So far, this approach seems to work well. At the same time, the author must devise his own methods to solve his problems. Lyee has a long history of development (some 20 years of sincere efforts, they say). The author does not think he can climb that high level in such a short period, but by making the presentation of inner logic in very traditional way, he thinks he can collaborate his works together with many researchers in the world.

Several examples of future research topics are:

- Removing unnecessary restrictions or hypotheses
- Augmentation of database handling
- Connection of generated application with web
- Systematic way of accumulating useful libraries

Acknowledgement

The author expresses his sincere gratitude to Dr. Fumio Negoro for his generousness to expose his deep thinking on Lyee. It is amazing how Dr. Negoro cut off unnecessary traditional thinking and presented really original ideas. Mr. Shigeaki Tomura always welcomed the author and answered the author's impudent questions and corrected the author's misunderstanding either at the meeting or via e-mail. The author also thanks Mr. Yozo Takeda for real problems experienced in practical businesses.

During experiment of Java coding Mr. Daisuke Usui helped the author in debugging Java program. Without his working knowledge, the author could not finish this paper.

References

- [1] F. Negoro, Principles of Lyee Software. Proceedings of 2000 International Conference on Information Society in the 21st Century (IS2000), 2000, pp. 441-446.
- [2] F. Negoro, Intent Operationalisation for Source Code Generation. Proceedings of the World Multiconference on Systemics, Cybernetics and Informatics (SCI2001), Volume XIV, Computer Science and Engineering Part II, 2001, pp. 496-503.
- [3] Issam A. Hamid and F. Negoro, New Innovation on Software Implementation Methodology for 21st Century - What Software Science can Bring to Natural Language Processing. Proceedings of the World Multiconference on Systemics, Cybernetics and Informatics (SCI2001), Volume XIV, Computer Science and Engineering Part II, 2001, pp. 487-489.
- [4] F. Negoro, The Predicate Structure to Represent the Intention for Software. Proceedings of the ACIS 2nd International Conference on Software Engineering, Artificial Intelligence, Networking & Parallel/Distributed Computing (SNPD'01), 2001, pp. 985-992.
- [5] F. Negoro, A Proposal for Requirement Engineering. Proceedings of the 5th East-European Conference on Advances in Databases and Information Systems (ADBIS2001), Volume 2, Tutorials, Professional Communications and Reports, 2001, pp. 7-18.
- [6] F. Negoro, Method to Determine Software in a Deterministic Manner. Proceedings of the International Conferences on Info-tech & Info-net (ICII2001), Conference D, pp. 124-129, 2001.
- [7] R. Hotaka and M. Tsubaki, Sentential Database Design Method, Proceedings of Advanced Database System Symposium (Information Processing Society of Japan), Dec. 9-10, 1981, pp. 53-60 (available from <http://shakosv.sk.tsukuba.ac.jp/~hotaka/paper/SDDM.pdf>).

R. Horaka / Engineering Characteristics of Line Program



Legend

```
noOfTrueWords=0 //this counter is to detect the change of the states
```

noOfTrueWords=noOfTrueWords+2 resulting 2

- *4 Deduction.W03.AccountNo=true
 Deduction.W03.Deduction=true
 noOfTrueWords=noOfTrueWords+2 resulting 4 (counting the words Deduction.W03.AccountNo and Deduction.W03.Deduction)
 Deduction.W03.Balance remains false because Login.W02.PassWord is empty
 set the flag showing the existence of a scheduled
- *5 since W04.prvNoOfTrueWrds(=0)<noOfTrueWords(=4)
 repeat currentBS(=Deduction)
 Disable guiDeduction
- *6 Deduction.W04.prvNoOfTrueWrds=noOfTrueWords(=4)
 repeat execution of words in this W04 until these words become stable
 noOfTrueWords=noOfTrueWords+2 resulting 6 (counting the words Deduction.W04.AccountNo and Deduction.W04.Deduction)
 Output the content of AccountNo and Deduction
- *7 as for Deduction.W03.AccountNO and Deduction.W03.Deduction, do nothing because both are true
 Deduction.W03.Balance remains false because Login.W02.PassWord is empty (the same check as in *4 again
 sinceDeduction.W03.Balance=false)
- *8 since Deduction.W04.prvNoOfTrueWrds<noOfTrueWords, Deduction is not stable.
 repeat currentBS(=Deduction)
- *9 Deduction.W04.prvNoOfTrueWrds=noOfTrueWords(=6)
- *10 as for Deduction.W03.AccountNO and Deduction.W03.Deduction, do nothing because both are true
 Deduction.W03.Balance remains false because Login.W02.PassWord is empty (the same check as in *4 again
 sinceDeduction.W03.Balance=false)
- *11 since Deduction.W04.prvNoOfTrueWrds=noOfTrueWords, Deduction is stable.
 since the flag showing the existence of a scheduled is set
 it is necessary to search for BS's as follows:
 Search for a word in Deduction.W03 which is false (in this case Deduction.W03.Balance fits this condition)
 if the derivation part of the corresponding W04 word
 (in this case max(0, ColVal(Login.W02.Password, "Account", "Balance")-Deduction.W02.Deduction))
 refers to some BS that have not appeared in BS's starting from rootBSPosition until currentBSPosition in BSStack,
 then stack that newly found BS on BSStack
 repeat the above, thus finding Login in this case
 currentBSPosition=currentBSPosition+1 resulting 2

store Deduction.prvNoOfTrueWords(=6) as the value of BSprvNoOfTrueWrds of Deduction in the BSStack
stack Login together with new level(=1)

currentBS=Login

reset the flag (of the current BS) showing the existence of a scheduled

setVisible and enable guiLogin

transfer to currentBS(=Login)

- *12 Login.W04.prvNoOfTrueWrds=noOfTrueWords(=6)
- *13 wait the input
- *14 Login.W02.Password=123
noOfTrueWords=noOfTrueWords+1 resulting 7
- *15 Login.W03.Password=true
noOfTrueWords=noOfTrueWords+1 resulting 8 (counting the word Login.W03.Password)

- *16 since Login.W04.prvNoOfTrueWrds<noOfTrueWords, Login is not stable.
repeat currentBS(=Login)
Disable guiLogin
- *17 Login.W04.prvNoOfTrueWrds=noOfTrueWords(=8)
repeated execution of words in this W04 until these words become stable
in this case, the value of Login.W04.Password is *****
noOfTreWords=noOfTrueWords+1 resulting 9 (counting the word Login.W04.Password)
Output the content of Password
- *18 as for Login.W03.Password, do nothing because it is true
- *19 since Login.W04.prvNoOfTrueWrds<noOfTrueWords, Login is not stable.
repeat currentBS(=Login)
- *20 Login.W04.prvNoOfTrueWrds=noOfTrueWords(=9)
- *21 as for Login.W03.Password, do nothing because it is true
- *22 since Login.W04.prvNoOfTrueWrds=noOfTrueWords, Login is stable.
and there are no words <w> such that Loin.W03.<w>=false, therefore
it is not necessary to search for another BS. just return to the point from where Login was called:
Unstack BSStack to get Deduction again (i.e., currentBSPosition=currentBSPosition-1)
Deduction.W04.prvNoOfTrueWrds=6
currentBS=Deduction

BSStack		
BS	level	BSprvNoOfTrueWrds
Deductio	0	6
Login	1	

continue to check if false value of some word in Deduction.W03 can change to true

in this case, the value of Deduction.W03.Balance changes to true because Login.W02.Password has value now

noOfTrueWords=noOfTrueWords+1 resulting 9

since Deduction.W04.prvNoOfTrueWrds<noOfTrueWords, Deduction is not stable.

repeat currentBS(Deduction)

*23 Deduction.W04.prvNoOfTrueWrds=noOfTrueWords(=9)

repeat execution of words in this W04 until these words become stable

noOfTrueWords=noOfTrueWords+1 resulting 10 (counting the word Deduction.W04.Balance)

Output the content of AccountNo, Deduction and Balance

*24 as for Deduction.W03.AccountNO, Deduction.W03.Deduction and Deduction.W03.Balance, do nothing because all of these are true

*25 since Deduction.W04.prvNoOfTrueWrds<noOfTrueWords, Deduction is not stable.

repeat currentBS(=Deduction)

*26 Deduction.W04.prvNoOfTrueWrds=noOfTrueWords(=10)

as for AccountNo, Deduction and Balance, do nothing because all of these have non-empty values and

noOfTrueWords remains unchanged

*27 as for Deduction.W03.AccountNO, Deduction.W03.Deduction and Deduction.W03.Balance, do nothing because all of these are true

*28 since Deduction.W04.prvNoOfTrueWrds=noOfTrueWords, Deduction is stable.

there remains nothing to process. Stop processing.

Ontologies and Information Systems: the Marriage of the Century?

Domenico M. PISANELLI, Aldo GANGEMI, Geri STEVE

CNR – ISTC, Viale Marx 15, 00137, Rome, Italy

Abstract. Although is recognized that ontologies may help building better and more interoperable information systems, there is skepticism on the real impact they may have in the future. We believe that ontologies will succeed in the information system arena and no systems will ever be designed without an ontological approach. In this paper we demonstrate the effectiveness of the ontological approach by illustrating three case studies. We show how an ontological framework is able to support semantic interoperability in the domain of fishery, then we present the role of ontologies for managing clinical guidelines and finally we sketch up an ontological analysis aimed at the interoperability of genetics databases.

1. Introduction

Many people today acknowledge that ontologies may help building better and more interoperable information systems. On the other hand, many others are skeptical about the real impact that ontologies - apart from the academic world - may have on the design and maintenance of working information systems.

Our claim is that ontologies will eventually succeed in the information system arena, the "marriage" will be happy and no computerized systems in this century will ever be designed without an ontological approach.

If "no man is an island", "no system is an island" anymore: data and knowledge integration (could we say "globalization"?) are no longer an optional, but a clear necessity. In fact, the overwhelming amount of information stored in various data repositories - including those available over the web - emphasizes the relevance of knowledge integration methodologies and techniques to facilitate data sharing. The need for such integration has been already perceived for several years, but telecommunications and networking are quickly and dramatically changing the scenario.

However, the ever-increasing demand of data sharing has to rely on a solid conceptual foundation in order to give a precise semantics to the terabytes available in different databases and eventually traveling over the networks. The actual demand is not for a unique conceptualization, but for an unambiguous communication of complex and detailed concepts (possibly expressed in different languages), leaving each user free to make explicit his/her conceptualization.

Ontology is the best candidate to face these problematics. Apart from its definition in the philosophical context - where it refers to the subject of existence - *ontology* in our context is "a partial specification of a conceptualization"[1], whereas Sowa proposed the following definition influenced by Leibniz [2]:

“The subject of ontology is the study of the categories of things that exist or may exist in some domain. The product of such a study, called an ontology, is a catalog of the types of things that are assumed to exist in a domain of interest D from the perspective of a person who uses a language L for the purpose of talking about D. [...]”

Actually there is some disagreement on what is an ontology. Some admit informal descriptions and hierarchies, only aimed at organizing some uses of natural language; others require that an ontology be a *theory*, i.e. a formal vocabulary with axioms defined on such vocabulary, possibly with the help of some axiom schema, as in description logics (for a position see Hayes [3]).

The various opinions in the research world enrich the scientific debate and are clear symptoms of cultural vivacity. Anyhow, the aim of this paper is not to review the ontology of ontology definitions, but to emphasize their strategic role in information systems design. Rather than raising up abstract claims, our objective is to demonstrate the effectiveness of an ontological approach by illustrating three case studies.

In the next paragraph we show how an ontological framework is able to support semantic interoperability among different information systems in the domain of fishery. The third paragraph presents the role of ontologies for effective and unambiguous dissemination of clinical guidelines. The fourth one sketches up an ontological analysis aimed at the interoperability of genetics databases.

2. The Fishery Ontology Service

Specialized distributed systems are state-of-the-art of current information systems architecture. Developing specialized information resources in response to specific user needs and area of specialization has its own advantage in fulfilling the information needs of target users. However, such systems usually use different knowledge organization tools such as vocabularies, taxonomies and classification systems to manage and organize information.

Although the practice of using knowledge organization tools to support document tagging (thesaurus-based indexing) and information retrieval (thesaurus-based search) improves the functions of a particular information system, it is leading to the problem of integrating information from different sources due to lack of semantic interoperability that exists among knowledge organization tools used in different information systems.

The different fishery information systems and portals that provide access to fishery information resources are one example of such scenario. In this paragraph we show the proposed solution to solve the problem of information integration in fishery information systems. The proposal shows how a fishery ontology that integrates the different thesauri and taxonomies in the fishery domain could help in integrating information from different sources be it for a simple one-access portal or a sophisticated web services application.

2.1 The local scenario

Fishery Ontology Service (FOS) is a key feature of the Enhanced Online Multilingual Fishery Thesaurus, a project aimed at information integration in the fishery domain. It undertakes the problem of accessing and/or integrating fishery information that is already partly accessible from dedicated portals and other web services.

The organisations involved in the project are: FAO Fisheries Department (FIGIS), ASFA Secretariat, FAO WAICENT (GIL), the oneFish service of SIFAR, and the

Ontology and Conceptual Modelling Group at ISTC-CNR. The systems to be integrated are: the "reference tables" underlying the FIGIS portal [4], the ASFA online thesaurus [5], the fishery part of the AGROVOC online thesaurus [6], and the oneFish community directory [7].

The official task of the project is "to achieve better indexing and retrieval of information, and increased interaction and knowledge sharing within the fishery community". The focus is therefore on tasks (indexing, retrieval, and sharing of mainly documentary resources) that involve recognising an *internal structure* in the content of texts (documents, web sites, etc.). Within the semantic web community and the intelligent information integration research area (see for example [8] and [9]), it is becoming widely accepted that content capturing, integration, and management require the development of detailed, formal ontologies. In the following we present an outline of the FOS development and some hint of the functionalities that it carries out, thanks to the ontological approach.

2.2 *Coping with heterogeneous systems: the ontological approach*

An example of how formal ontologies can be relevant for fishery information services is shown by the information that someone could get if interested in "aquaculture". In fact, beyond simple keyword-based searching, searches based on tagged content or sophisticated natural-language techniques require some conceptual structuring of the linguistic content of texts.

The four systems concerned by this project provide this structure in very different ways and with different conceptual 'textures'. For example, the AGROVOC and ASFA thesauri put "aquaculture" in the context of different thesaurus hierarchies: according to AGROVOC the terms more specific than "aquaculture" are "fish culture" and "frog culture", whereas in ASFA they are "brackishwater aquaculture", "freshwater aquaculture", "marine aquaculture". Two different contexts relating respectively to species and environment point of view.

With such different interpretations of a term, we can reasonably expect different search and indexing results. Nevertheless, our approach to information integration and ontology building is not that of creating a homogeneous system in the sense of a reduced freedom of interpretation, but in the sense of navigating alternative interpretations, querying alternative systems, and conceiving alternative contexts of use.

To do this, we require a comprehensive set of ontologies that are designed in a way that admits the existence of many possible pathways among concepts under a common conceptual framework. This framework should reuse domain-independent components, be flexible enough, and be focused on the main reasoning schemas for the domain at hand.

Domain-independent, *upper* ontologies characterise all the general notions needed to talk about economics, biological species, fish production techniques; for example: *parts*, *agents*, *attribute*, *aggregates*, *activities*, *plans*, *devices*, *species*, *regions of space or time*, etc. On the other hand, the so-called *core* ontologies characterise the main conceptual habits (schemas) that fishery people actually use, namely that certain plans govern certain activities involving certain devices applied to the capturing or production of a certain fish species in certain areas of water regions, etc.

Upper and core ontologies [10,11] provide the framework to integrate in a meaningful and *intersubjective* way different views on the same domain, such as those represented by the queries that can be done to an information system.

2.3 *Ontology integration and merging*

Once made clear that different fishery information systems provide different views on the domain, we directly enter the paradigm of *ontology integration*, namely the integration of schemas that are arbitrary logical theories, and hence can have multiple models (as opposed to database schemas that have only one model) [12]. As a matter of fact, thesauri, topic trees and reference tables used in the systems to be integrated could be considered as *informal* schemas conceived to query semi-formal or informal databases such as texts and tagged documents.

In order to benefit from the ontology integration framework, we must transform informal schemas into *formal* ones. In other words, thesauri and other terminology management resources must be transformed into (formal) ontologies. To perform this task, we apply the techniques of three methodologies: OntoClean [11], ONIONS [13], and OnTopic [14] which are described in detail in the literature.

OntoClean consists of principles for building and using upper ontologies for core and domain ontology analysis, revision, and development. The OntoClean methodology is based on highly general ontological notions drawn from philosophical ontology, especially from what is now called “analytic metaphysics”, and it is general enough to be used in any ontology effort, independently of a particular domain.

As claimed by the authors: “The OntoClean methodology provides a formal, consistent and straightforward way to explain some of the most common misunderstandings in conceptual modeling regarding the taxonomic or subsumption relation.”[11]. In fact, those taxonomies feeding the first steps of an ontological analysis very often confuse subsumption with instantiation (e.g.: “John” is an instance of “Human” whereas the class “Humans” is subsumed by the class “Mammals”). Not to forget that another frequent feature of bad taxonomies is the systematic confusion between “part_of” and “is_a” (subsumption) relationships (e.g.: “engine” part_of “car”; “car” is_a “vehicle”).

ONIONS (“ONtological Integration Of Naïve Sources”) is a set of methods for enhancing the informal data of terminological resources to the status of formal ontological data types. OnTopic is about creating dependencies between topic hierarchies and ontologies. It contains methods for deriving the elements of an ontology that describe a given topic, and methods to build ‘active’ topics that are defined according to the dependency of any individual, concept, or relation in an ontology.

3. *Ontologies and Clinical Guidelines*

Guidelines for clinical practice are being introduced in an extensive way in more and more different fields of medicine [15,16]. They have the goal of indicating the most appropriate decisional and procedural behavior optimizing health outcomes, costs and clinical decisions.

Guidelines can be expressed in a textual way as recommendations or in a more formal and rigid way as protocols or flow diagrams. In different contexts they can be either a loose indication for a preferred set of choices or they can be considered a normative set of rules.

Clinical practice guidelines are seen as a tool for improving the quality and cost-efficiency of care in an increasingly complex health care delivery environment. It has been proved that adherence to plans may reduce cost of care up to 25% [17].

However the overwhelming number of guidelines available makes it difficult to select the right one. Just to give an idea of the figures, it is reported that there are 855 different guidelines for British GPs ranging from a single page to small booklets of more than 15 pages [18].

Computerization may increase the effectiveness of both the information retrieval of guidelines and the delivery of guideline-based care. In an optimal scenario they are integrated with the information systems operational at the point of care. The full potentialities of computerized systems can be exploited in such an environment where different processes are executed in parallel on several patients. In this context such systems must be able to retrieve the updated situation of every patient, as well as to give an overall report on the ward, freeing the physicians to concentrate more on clinical decisions. Keeping track of the parallel activities performed, they should avoid unnecessary duplication of tasks and prevent possible omissions.

3.1 The unambiguous representation of guidelines

Several research projects deal with the computer representation and implementation of guidelines. The scenario is evolving from stand-alone workstations to telematics applications that - utilizing e.g. the Internet - not only support the use of guidelines, but also enable their development and dissemination.

Such a knowledge sharing requires the definition of formal models for guidelines representation. The models should have a clear semantics in order to avoid ambiguities.

The role of ontologies is that of making explicit the conceptualizations behind a model. The definition of ontologies - i.e. the formal description of the entities to which a model makes a commitment and of the relations holding among the entities - is the groundwork for making a standard model acceptable and sharable. An ontology library is not normative, but allows an inter-subjective, explicit and formal agreement on the semantics of the primitives of a model, by referring to more generic primitives (generic theories).

We believe that such an approach can facilitate the standardization process by allowing an explicit mapping in a formal ontology of the concepts represented in the heterogeneous models proposed so far.

In our ontology guidelines are distinguished in "paper guidelines" and "web guidelines". Some common concepts - like "author" - pertain to both of them, whereas "URL" and "last-checked" are peculiar of the web guidelines. They are also categorized in five different kinds, as defined in the Guideline Interchange Format standard (GLIF) [19]: "guideline for care of clinical condition", "screening and prevention", "diagnosis and prediagnosis management of patients", "indications for use of surgical procedures", "appropriate use of specific technologies and tests". Such classification is furtherly specialized by us: for example a "guideline for care of clinical condition" may be a "therapy assessment", a "pharmacologic therapy" or a "disease management".

As far as the formal representation of guidelines is concerned, our ontology integrates some of the most relevant modeling efforts so far produced: notably PROforma [20], EON [21], Asbru [22] and GLIF [19]. It is also an evolution of a model previously defined in the context of the SMART system [23]. A "guideline" is a kind of "plan" which is a method of a "procedure", and it is represented by a "flowchart" (for more details see [24,25]).

The concept of "flowchart" pertains to the "diagrams" ontology. It is defined as a set of nodes and edges like an ordinary graph with some restrictions. Every flowchart has a first and a last node, only four kinds of nodes are allowed: single nodes, branching

nodes, synch nodes and cycle nodes. Moreover the flowchart ontology allows for recursion, i.e. a node may be expanded into a flowchart.

3.2 *The ontology of planning*

The ontology of clinical guidelines accounts for the structural part of a guideline, but no semantics is attached to it. The semantics of the actions involved pertains to the *planning* ontology, where simple nodes represent elementary actions and branching nodes enquiries and decisions. The recursion allowed in the flowchart domain, where a node of a flowchart may be expanded into a flowchart, is isomorph to the planning ontology, where an elementary action may be refined into a plan.

We believe that in our model it is appropriate to capture the distinction between the structural part of a guideline, represented by the flowchart, and its semantics, represented by the plan. A third level is that of the procedure, i.e. what is actually performed.

This ontology integrates some of the most relevant work in the guideline modeling field. It is GLIF-compliant, i.e. each concept defined in GLIF is represented in it (e.g. the "synch" node after parallelization of activities). It takes into account the ProForma task ontology which categorizes tasks into: actions, enquiries and decisions and allows recursive definition of them (a plan is made of tasks, a task may be a plan).

It has been proven that the introduction of guidelines can significantly decrease the costs of care and therefore they are a "hot topic" in the agenda of health care professionals. Guidelines are mushrooming and computers can help in retrieving them and can give assistance during their execution.

However such a widespread diffusion poses new problems, not only in terms of credibility and acceptability, but also concerning non-ambiguity in knowledge dissemination. Formal models with a clear semantics should be defined in order to represent guidelines and facilitate their diffusion.

4. **Ontology for the Interoperability of Genetics Databases**

We are witnessing the unification of biology, since both biochemists and genetists now recognize a single universe of genes and proteins, and such unification is made possible also by the ever-increasing availability of the sequences of entire genomes. Such an availability should rely on solid conceptual foundations in order to give a precise semantics to the data present in the different genome databases and accessible over the networks.

The conceptualization task - which is the groundwork for solid foundations - is not an easy one to be achieved, since a deep analysis of the structure and the concepts of terminologies is needed. Such analyses can be performed by adopting an *ontological* approach for representing terminology systems and for integrating them in a set of ontologies.

We performed an ontological analysis of the Metathesaurus™ [26], a terminology data-bank developed in the context of the Unified Medical Language System (UMLS) project by the U.S. National Library of Medicine [27,28]. It collects millions of terms belonging to the most important nomenclatures and terminologies defined in the United States and in other countries too. About 700,000 preferred terms, named "concepts", have been chosen as representative of a set of synonyms and lexical variants in different languages. It is probably the largest repository of terminological knowledge in medicine. Recently we extended ontological analysis in the genetics field.

As a case study, we investigated on the molecular function ontology defined by the Gene Ontology Consortium (<http://www.geneontology.org>) [29]. We implemented a wrapper translating from the XML ontology definition into LOOM, a formalism suitable for automatic classification [30].

The ontological analysis put in evidence the necessity of refining some assumptions made by the Gene Ontology developers. For example, metonymy is often used, since both enzymes and their functions are used in the same taxonomy.

5. Conclusions

Heterogeneity of information in data bases schemata or in other semi-formal information repositories is due essentially to the different conceptualizations, the different intended meanings of the terms which constitute the information in the repository. Such inherent polysemy of terminological information is reflected by widespread polysemous phenomena within existing terminologies.

Usually terminological sources shows the following features:

- *Lack of axioms*: for example, ICD10 shows nude taxonomies, without axioms or even a natural language gloss.
- *Semantic imprecision* (cycles, relation range violation, etc.): for example, the semantic network used as the top-level of the UMLS Metathesaurus includes a set of templates for its taxonomy, but the semantics of such templates is not defined at all: after careful analysis, the best that we could do is considering UMLS templates as default axioms.
- *Ontological opaqueness* (lack of motivation for choosing a certain predicate, or lack of reference to an explicit, axiomatized generic ontology, or at least to a generic informal theory): for example, systems in which concepts and relations in the top-level are non-axiomatized and undocumented: they may appear to have been chosen with disregard of formal ontology: possibly no trace of mereological, topological, localistic, dependence notions is retrievable.
- *Linguistic awkwardness in naming policy*: for example, systems in which purely formal architecture considerations originate a lot of redundancy and cryptic relation and concept names.

Ontology integration may act as a reference activity for information integration architectures and standardization work. Our experience has proved that the ontologies produced by means of the ONIONS methodology support:

- *Formal upgrading* of terminology systems: term classification and definitions are now available in a common, expressive formal language;
- *Conceptual explicitness* of terminology systems: (local) term definitions are now available, even though the source does not include them explicitly;
- *Conceptual upgrading* of terminology systems: term classification and definitions are translated so that they can be included in an ontology library which has a subset constituted of adequate generic ontologies;
- *Ontological comparability*, since pre-existing ontology libraries pertaining to different fields are largely employed.

In conclusion, we point out the following important features of ontologies:

- Semantic explicitness.
- An explicit taxonomy.
- Explicit linkage to concepts and relations from generic theories.
- Absence of polysemy within a given formal context.

- Modularity of contexts.
- Some minimal axiomatization to detail the difference among sibling concepts.
- A good naming policy.
- Rich documentation.

Our research aims at showing that integration of heterogeneous information systems can take advantage from the framework of formal ontology. We recognize that, in a software engineering perspective, many drawbacks are still present in ontology-based information systems.

Tools for efficiently supporting an ontology-based system design are not reliable enough for being employed in industrial practice. However we do believe that in a near future better engineered instruments will be available and the advantages of ontology-based design and integration of information systems will be tangible.

To this aim, the ontology design can profit also from the peculiar *Lyee* approach [31]. Even an ontology, in its essence, is a piece of software, and *Lyee* is a powerful and innovative software engineering methodology and tool for implementing software. It has been shown that such a tool allows an implementation faster than by using traditional methods.

Exploring the potentially fruitful synergies between the "classic" ontological analysis method and the revolutionary *Lyee* approach, will be a prominent issue in our future research.

References

- [1] Guarino N (ed.), *Formal Ontology in Information Systems*, Amsterdam, IOS-Press, 1998.
- [2] Sowa J, communication to the *ontology-std* mailing list, 1997.
- [3] Hayes P, note on the meaning of "ontology". <http://ksl-web.stanford.edu/email-archives/srkb.messages/647.html>.
- [4] <http://www.fao.org/fi>
- [5] <http://www4.fao.org/asfa>
- [6] <http://www.fao.org/agrovoc>
- [7] <http://www.onefish.org>
- [8] <http://www.ontoweb.org>
- [9] <http://www-2.cs.cmu.edu/afs/cs.cmu.edu/project/theo-6/web-agent/www/i3.html>
- [10] Gangemi A, Guarino N, Masolo C, Oltramari A.: Understanding Top-Level Ontological Distinctions, in: H. Stuckenschmidt (ed), *Proceedings of the IJCAI 2001 Workshop on Ontologies and Information Sharing*.
- [11] Guarino N, Welty Ch. Evaluating Ontological Decisions with Ontoclean. *Communications of the ACM*, 2002, vol.45 (2): 61-65.
- [12] Calvanese D, De Giacomo G, Lenzerini M.: A Framework for Ontology Integration. *Proceedings of 2001 Int. Semantic Web Working Symposium (SWWS 2001)*.
- [13] Gangemi A, Pisanelli DM, Steve G.: An Overview of the ONIONS Project: Applying Ontologies to the Integration of Medical Terminologies. *Data and Knowledge Engineering*, 1999, vol.31, pp. 183-220 (1999)
- [14] Gangemi A, Pisanelli DM, Steve G.: The OnTopic Methodology for Supporting Active Catalogues with Formal Ontologies. *ISTC-CNR-OCMG Internal Report iii-01* (2001)
- [15] Woolf SH. Practice Guidelines, a New Reality in Medicine: Impact on Patient Care. *Arch Intern Med*, 1993; 153: 2646-2655.
- [16] Grimshaw JM, Russell IT. Effect of Clinical Guidelines on Medical Practice: a Systematic Review of Rigorous Evaluations. *Lancet*, 1993; 342: 1317-1322.
- [17] Clayton PD, Hripcsak G. Decision support in healthcare. *Int. J. of Bio-Medical Computing*, 1995; 39: 59-66.
- [18] Hibble A., Kanka D., Penheon D., Pooles F: Guidelines in general practice: The new Tower of Babel? *British Medical Journal*, 1998; 317: 862-3.
- [19] Ohno-Machado L et al. The Guideline Interchange Format. *Journal of American Medical Informatics Association*, 1998; 6.

- [20] Fox J, Johns N, Rahmzadeh A. Disseminating Medical Knowledge: The PROforma Approach *Artificial Intelligence in Medicine*, 1998; 14.
- [21] Musen MA, Tu SW, Das AK, Shahar Y. EON: A component-based approach to automation of protocol-directed therapy. *JAMIA*, 1996; 3.
- [22] Shahar Y, Miksch S, Johnson P. The Asgaard project. *Artificial Intelligence in Medicine*, 1998; 14.
- [23] Pisanelli DM, Consorti F, Meriardo P. SMART: A System Supporting Medical Activities in Real Time, *Proc. Medical Informatics Europe*, 1997.
- [24] Pisanelli DM, Gangemi A, Steve G, "Towards a Standard for Guideline Representation: an Ontological Approach", *Journal of American Medical Informatics Association*, vol.6 S4, pp. 906-910, 1999.
- [25] Pisanelli DM, Gangemi A, Steve G, "The Role of Ontologies for an Effective and Unambiguous Dissemination of Clinical Guidelines", in R Dieng, O Corby (eds.), "Knowledge Engineering and Knowledge Management. Methods, Models, and Tools", Berlin, Springer-Verlag, pp. 129-139, 2000.
- [26] Humphreys BL, Lindberg DA, "The Unified Medical Language System Project", *Proceedings of MEDINFO 92*, Amsterdam, Elsevier, 1992.
- [27] Pisanelli DM, Gangemi A, Steve G, "An Ontological Analysis of the UMLS Metathesaurus", *JAMIA*, vol. 5 S4, pp. 810-814, 1998.
- [28] Pisanelli DM, Gangemi A, Steve G, "A Medical Ontology Library that Integrates the UMLS Metathesaurus™", *Lecture Notes in Artificial Intelligence 1620*, Berlin, Springer Verlag, pp. 239-248, 1999.
- [29] The Gene Ontology Consortium, "Gene Ontology: tool for the unification of biology", *Nature Genetics*, vol.25, pp.25-29, 2000.
- [30] MacGregor RM, "A Description Classifier for the Predicate Calculus" *Proceedings of AAAI 94, Conference*, 1994.
- [31] Negoro F, "Principle of Lyee software", *Proceedings of 2000 International Conference on Information Society in 21st Century (IS2000)*, pp.441-446, 2000.

The Key Features of the LyeeAll Technology for Programming Business-like Applications¹

Victor MALYSHKIN

*Supercomputer Software Department (SSD),
Institute of Computational Mathematics and Mathematical Geophysics,
Russian Academy of Sciences, Novosibirsk, RUSSIA*
malysh@ssd.sccc.ru, <http://ssd.sccc.ru>

Abstract. The key features of the new commercially successful technological system LyeeAll are considered. The LyeeAll system is oriented to the development of business-like application software. Its key features are the automation of the development of the requirement definition; the automatic program generation from requirement definitions, in order to eliminate several important steps of the software development process like modularization, design of data structures and programs, program development and maintenance, etc.; standardization of the data and control structures of a generated program; standardized user interface of application programs.

1. Introduction

The key features of the new commercially successful software technological system LyeeAll are considered. The author visited the Institute of Computer Based Software Methodology and Technology (ICBSMT, Tokyo, Japan) where the new methodology Lyee [1] of the software development has been created. The key concept of the methodology is to support the software development process at the earliest possible stage. The description of the Lyee methodology is given using the philosophical notions like meaning, consciousness, cognition, logical atom, existence and others. Being not defined mathematically these terms seem to be vague for the author. For this reason the Lyee methodology is not considered here.

The LyeeAll system is one of the possible implementations of the Lyee approach. The LyeeAll was especially designed in order to support the development of business-like applications. Its most attractive features are supporting the process of the requirement definitions (RD) development and automatic program generation directly from the RD. As a result, several serious stages of the software design and software development like modularization; design of algorithms, data structures and programs; program development and maintenance are practically eliminated. That is why this system attracts the attention among the numerous others.

Composition of different approaches to program construction such as single assignment, program generation with macro generator, standardized data and control structures of a generated program have provided good practical success of the LyeeAll system and made it apt to the business-like applications.

¹ This paper hereof is contributed to the Lyee International Collaborative Research Project sponsored by Catena Corp. and The Institute of Computer Based Software Methodology and Technology.

In order to know exactly what is really being done by the LyeeAll system and to look at how the design of a problem solution is performed, the solution of a certain business application problem was done and then the generated code was analyzed. This provided both the extraction of the objective information on the LyeeAll and the possibility to make the general analysis of the technology. The most important features and limitations of the LyeeAll system are presented below. The demonstrative materials were provided by the ICBSMT.

2. General description

A general scheme of the LyeeAll can be found in Fig.1. End user works with *Interface subsystem* that extracts the RD from the user. This is done by the active obtrusion of the LyeeAll style of the RD construction and representation on the end user. LyeeAll demands to represent the RD as sets of input and output variables (similar to the declarative style of the description of the functional specification in a system of program synthesis), which is to be specified in detail for the proper implementation.

All the variables of a problem description are represented as windows and buttons of the *design screens*, and for each output window a *fragment* of computation (procedure, module), issuing this output, is developed. The output variable of the design screen can be used as input variable in computing other output variables. Every window represents either a simple variable or an aggregative variable or a record of a file. This results that finally an application algorithm is represented as a finite or a potentially infinite set of functional terms [4], whose operations should be developed by the end user. This predefines as advantages as limitations of a program generated under LyeeAll system.

The formed RD specifies the desired algorithm of the problem solution. The RD is input into *Source code generation subsystem*. Source code generator is actually a macro generator. It uses the library of templates for several programming languages (C, Pascal, Visual Basic for now). These templates are really specifically designed macro definitions. According to selected templates the source code, implementing the specified algorithm, can be generated in C, Pascal or Visual Basic languages respectively.

The user interface of the generated program is also implemented as a screen with windows and buttons. This screen interface is generated with Visual Basic or Delphi.

3. Key features of the LyeeAll system

A number of key features of the LyeeAll provided its success. But the same features are responsible for the most serious limitations of the LyeeAll system. Each of them is more or less known and was earlier used in different technologies of programming. These features are shortly considered and demonstrated below.

Standardization and regularity of the design process. High level of standardization and regularity of all the stages of an application problem design is among the basic advantages of LyeeAll. What LyeeALL requires from the end user is the user's knowledge of what he/she really wants to develop. In order to clarify this, the interface subsystem of the LyeeALL strongly manages the user's work and force-feeds user a correct design decision. Certainly, this is a correct decision from the LyeeAll system viewpoint, but not generally.

Such a regularization practically eliminates the creative part of the design process. High level of standardization substantially reduces the duration of a problem design, as well as the possibility to commit an error and, in general, substantially reduces the thinking efforts. This provides high reliability, portability and maintainability of generated programs, their ability to evolution and many other important properties of a target

program. As result of a high level of standardization/regularity any design action under LyeeAll is more or less mechanical regular procedure.

Can all this ensure that expected RD will be exactly designed and that the desirable program will be generated? The answer is "No". But automatic program generation provides an easy way for RD corrections. On the other hand, the LyeeAll facilities ensure that in practice the desirable program can be finally specified and generated.

For business-like applications, the developed RD always provides the generation as good RD as good program. This is the consequence of linear complexity of the algorithms in this area.

Data standardized representation. Data description and data representation are strongly restricted. The project data are designed and represented as a set of screens. Each screen consists of elements of the two types: windows and buttons. The former are used to describe the objects of data, to input or to output the information, while the latter are intended for requests of a service. There is no other way to describe data. Each screen actually describes the structure of a document in a similar manner as is done for a database system. Fragments of computations for data processing are developed for each screen element and bound to it. The application system is always designed in such a way that the end-user is operating only with the screens of the project.

Automatic program generation. Chosen in LyeeAll the method of the RD development brings about the representation of the constructed algorithm as a set of functional terms (see in [2,3] the Kleene characterization of a computable function). The fragments of computations, developed in the design screen, represent the functional operations of the terms. The program generation is done by the macro generator, which implements a certain circuit of a tree (the graphical representation of a functional term).

Certainly the success of this approach is possible because peculiarities of the algorithm structure of the application area were taken into account. Among them the most important properties are the linear computational complexity of counting of a set of functional terms, representing an algorithm [2-4], and the linear complexity of a functional term circuit. These properties characterize business-like applications algorithms. In other words, the structure of application algorithms is simple enough. Being especially designed for the business-like application development, the LyeeAll system might be actually suitable for the development of the programs from other application areas, where the structure of algorithms is simple similar to the algorithms of the business application area.

A separate functional program is generated for processing of each element of a screen. A functional program includes processing of input data: issuing of output data: modification and calculation of data element. The state of any element (a button is either pressed or not, data are either input/typed in a window or not, etc.) is kept as a value of a control variable corresponding to the screen element. The control structure of any functional program is fixed. It is assembled out of the ready-made control fragments, named in LyeeAll *pallets*. The pallet W02 (macro definition W02) provides the control structure for the generation of that part of the target program, which is responsible for data input. The pallet W03 provides control template for functional computations, etc. Each functional program is generated in the single assignment mode. A value to any functional variable is assigned only once and is never re-assigned later. To every variable the separate memory location is assigned.

It seems that with a current ability to provide a big volume of different computer resources for program execution it now appears possible to use single assignment paradigm, that consumes huge volume of resources, even in solution of practical problems. Such a situation looks similar to the growing success of application of the brute force methods in solution of combinatorial problems, where with current high performance processors often it is more productive and technological way to accomplish full search than

to use sophisticated optimized search. The simplicity of an algorithm structure in business applications plays in this case important role: functional terms have a simple structure, procedures are not big (i.e., contain few statements), value of variables are consumed soon after their computation, accumulation of the valid variable values in the program does not occur.

Any LyeeAll generated program contains many statements that could be eliminated by a programmer. But the cost of a compact code is often too high (only remember the problem of program maintenance). On the other hand, any generated program is well structured.

The whole program is generated as a set of functional programs. The control over generated program is dynamically implemented. The current state of the program is described by the control variables, whose values define the direction of the control flow of the computation. Any functional program being invoked first checks whether it was already successfully executed or not. Before the completion any functional program leaves in a certain control variable the information on how it was completed: either successfully or not, should it be invoked later or not, etc.

For all these reasons, the structure of all the LyeeAll generated programs are similar to each other as twins.

High portability of the LyeeAll generated programs is provided by the choice of the library of templates for a certain programming language.

Maintenance of software elimination. The desirable program is constructed under LyeeAll directly from the RD. This means that the stage of program maintenance is absent here. To speak plainly, if a certain program is to be modified in order to satisfy new requirements it is easier to develop a new program (actually, to develop a new RD) than to modify the currently exploited program.

4. Example of the design of the digit clock screen

The LyeeAll system is successfully used for the development of business applications. But LyeeAll system can also be successfully applied in the areas where algorithms have the same linear complexity as business application algorithms (business-like applications). Let us look at the example of the digital clock design (fig. 2). Here it is necessary to develop a program that is to construct the image of a digit given in binary format (digit "3" in Fig.2, that is the value of the variable/window NOM).

A digit image is constructed out of seven elements/windows named Line1÷Line7. Each element is processed by a separate functional program. The whole program consists of processing of all seven elements that can be processed even in parallel. In Fig.2 the fragment of the functional program, processing Line6, is given. This program should compute a decision either to show the element Line6 or not, in order to construct the image of the digit "3". The variables W03.DIGIT.Line6, W03_RECALL_FLG are the examples of the control variables. The flow-chart of the pallet W03 in Fig.2 is the same for any functional program.

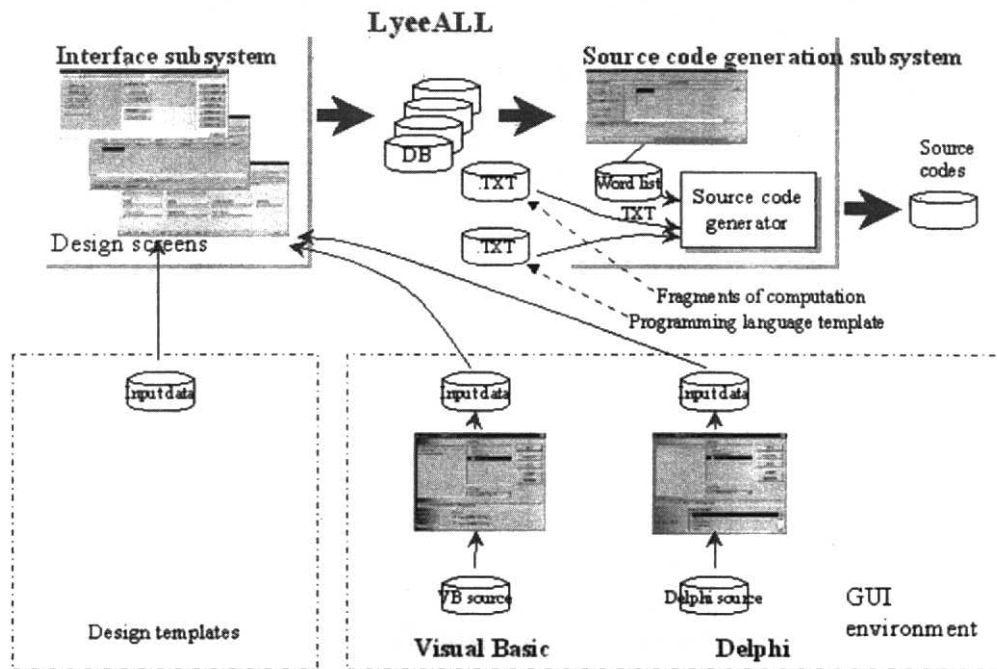


Fig 1

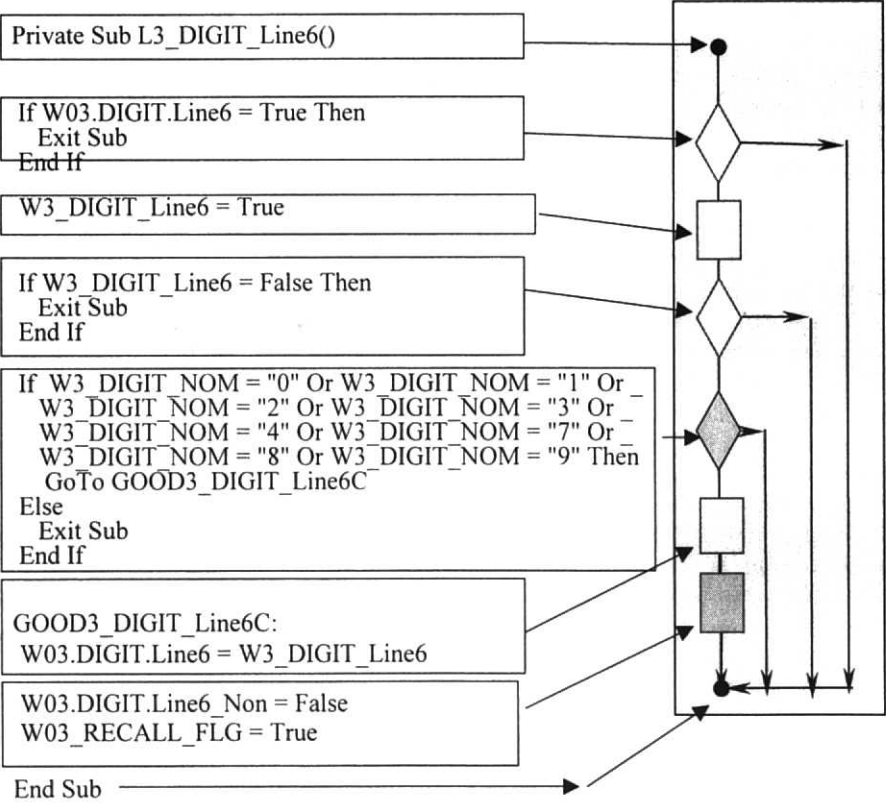
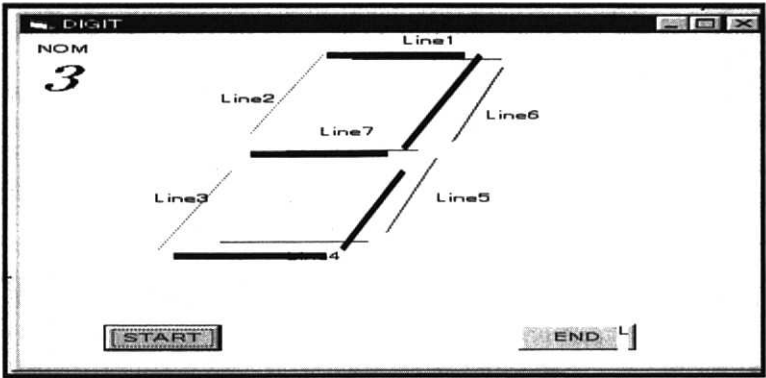


Fig. 2

5. Limitations of the LyeeAll

As any concrete implementation of a programming technology, LyeeAll system cannot avoid some critical remarks. The programs generated under LyeeAll can also be criticized from different viewpoints. As often happens advantages of a technological system generate its specific limitations.

The most substantial limitations of the LyeeAll system is that the only algorithm, implementing designed RD, is constructed. This means that computation optimization on the level of an algorithm construction is impossible.

The complexity of a constructed algorithm is strongly restricted. This limitation is incident to any system of program synthesis. Unfortunately, the structures of application algorithms in many application areas are not so simple as this is rightly for the algorithms in the area of business application. This is also not avoidable for now limitation of the LyeeAll system. Concentration on supporting the process of the RD development is the most promising feature of LyeeAll system as technology of programming. Theoretically, similar approach could be taken in the other application areas. One of the reasons why this cannot be done now is the absence of good algorithms for folding of the set of functional terms into more or less compact program of good performance that should also possess many other desirable/necessary properties.

Another limitation is the fixed algorithm of the code generation. It means that code optimization on the level of a program construction is also impossible. In particular, such a fixing will restrict the development of parallel programs whose flexibility and tunability are based on a set of algorithms, implemented by their code (static program), that describes (or should ideally describe in its turn) a set of executing programs (set of processes, dynamic program).

6. Conclusions

The LyeeAll system demonstrates a successful application in practice of such fundamental computational essences as algorithm representation by a set of functional terms and single assignment mode of programming.

There are different directions where LyeeAll technology might be developing. Such improvements should reduce the above listed limitations. Among them, for example, is the development of new algorithms of the code generation, etc. Such improvements might substantially extend the applicability of the LyeeAll.

Completing the paper there is a sense to say once more that successive application of LyeeAll system is possible only in the areas where all the algorithms of problem solution have linear computational complexity.

Finally I would like to thank Mr. Fumio Negoro for numerous discussions; Mr. Shigeaki Tomura for the presented demonstrative materials, his hard work to teach me LyeeAll technology and permanent friendship; generally all the staff members of the ICBSMT for their kind help during my stay in Tokyo; the junior researcher and programmer of the SSD Nikita Malyshkin for his valuable remarks and assistance.

References

1. *Official Reports & Documents of Lyee*, <http://www.lyee.co.jp>.
2. A.I. Mal'tsev, *Algorithms and Recursive Functions*, Wolters-Noordhoff, 1970
3. H. Rogers Jr., *Theory of Recursive Functions and Effective Computability*, McGraw-Hill, 1967
4. V.A.Valkovskii, V.E.Malyshkin. *Synthesis of Parallel Programs and Systems on the Basis of Computational Models*. Nauka, Novosibirsk, 1988. (In Russian. Sintez parallel'nykh programm i sistem na vychislitel'nykh modelyakh)

Software Process of the Insurance Application System by using Lyee Methodology

Volker GRUHN* , Raschid IJIOUI* , Dirk PETERS* , Clemens SCHÄFER*

Takuya KAWAKAMI[×] , Makoto ASARI[×]

** Fachbereich Informatik Lehrstuhl 10,*

Universität Dortmund, D-44221 Dortmund, Germany

[×] Faculty of Software and Information Science,

Iwate Prefectural University, Iwate, Japan

Abstract. With Lyee methodology many classical software engineering techniques cannot be used anymore. Especially classical project planning cannot directly be adopted because the software process is completely different to OOA/OOD. To understand the Lyee software process the development of an insurance application system is described and the process is modelled with LeuSmart.

1 Introduction

Lyee is not just a new programming language because Lyee does not only define syntax and semantic of programs. The whole software process is completely different to other known software processes. The intention of Lyee is to develop software in a shorter period of time with less costs. To prove the efficiency of Lyee empirical studies are problematic because the productivity of different programmers often varies more than a factor 10, although they use the same techniques to develop software. Comparing different software development techniques is even more difficult. Obviously, someone who is a professional OOA/OOD developer would develop much more efficiently with Together and Java than with Lyee and vice versa. To document the Lyee software process and its strengths and weaknesses accurately it is necessary to use a formal process language. LeuSmart is a FUNSOFT-net based process modelling tool, which we use to describe Lyee software processes.

In terminology of FUNSOFT nets a software process is a set of activities performed during software development. FUNSOFT nets are a Petri-net based formal language with a formal semantic. That means that it is possible to do formal software process documentation with FUNSOFT nets. Formal documentations of software processes are important for measuring, how successful software processes are. Since Lyee does not need many parts of traditional software engineering, the process is fundamental different. The problem is, that there is one software process for each software system that is developed, although Lyee has characteristics, that are part of Lyee like other characteristics are part of the waterfall model, the prototyping model or the spiral model. Characteristics of waterfall model, the prototyping model

and the spiral model are discussed in literature in detail during the past years, but this work still has to be done for Lyee to be widely accepted in software community. Today the FUNSOFT net is a well known modelling language for software processes documentation. This early work was extended during the following years by many researchers.

FUNSOFT nets are fully supported by a Software tool called LeuSmart. This makes FUNSOFT nets more powerful than ever for software processes documentation. The tool LeuSmart was developed by adesso AG and it was already used in a wide range of industrial projects. Another advantage for software process documentation is, that carefully documented and easy understandable software processes can be the basement for software process enhancements, project management etc. To get a better understanding, why FUNSOFT nets are so successful for software process documentation, we mention here a few highlights of the language and the tool. Of course this document is too short to give a full understanding.

2 Development Process of Lyee Applications

The idea of Lyee is to produce software without experiences and knowledge. Our goal is to examine this state on the basis of a process which we produced with LeuSmart. The goal of all software processes is the production of software. Software that works, software that is on time, software that is within budget, software that can be maintained, software that can be reused. If this goal is met while preserving the rights of the developers and customers, then the process is a success. There are many companies and government organizations that develop or maintain software to support their operations or their business products.

The development of software includes the creation of specification, design, source code, and testing. These different artefact interact with each other where a delay or defect in one affects the completeness of the others. This often results in a software product that is behind schedule, over budget, non-conforming to requirements and of poor quality. The result is that the company loses money or the government organization misuses taxpayer's money either through budget overruns or decreased user and customer satisfaction. Controlling and improving the process used to develop software is seen as the remedy to these problems.

An important question for industry and government is what are the benefits of investing resources to improve the Organization's Process Maturity. The long-term benefits of high process maturity are software delivered on time, within budget, within customer requirements, and of high quality. An important benefit would be the effect it has on productivity. The goal of this research is the discovery of the Lyee process quantified effect that Process Maturity has on software development effort and the modelling approach used to isolate the effects of Process Maturity on effort. Understanding Process Maturity's influence on effort within the context of other factors provides a trade-off analysis capability that can be used to lower the effort required to produce a software product. The modelling approach can be used in other areas of Software Engineering. The following development process is an organization of the development activities, described at a level of details that can be directly used as a model for instantiating the activities of a specific project for LyeeAll2 software development. For our development, we adhered to this process.

When modelling Lyee software processes, it is important to see, whether activities can be executed in parallel or only in a sequence, if there are alternative ways or iterations. This has a fundamental impact on the total time of the software production. It is not possible to know the time and sources needed for producing exactly simulation results. The best way to get these information is to watch an industrial software process very carefully in order to get good input for the software process model. In the following we represent the software development process of Lyee (see **Figure 1**).

Summary of Lyee's Development Process

Items of work are as follows:

1. Create Alias using the BDE Administrator
2. Registration of Data to LyeeAll2
 - 2.1 Database Login
 - 2.2 Project
 - 2.3 System
 - 2.4 Defined/Defined Details
 - 2.5 Process Route Diagram
 - 2.6 Pallet ID
 - 2.7 Pallet Defined
 - 2.8 Action Vector
 - 2.9 Logical Definition
3. Generate Source Code
4. User verification phase and user training (see Figure 2)

3 Development Environment

Lyee presents two sides: the theoretical component and its concretization in software tools (presently, LyeeALL2 and LyeeMAX). The former aspect is often presented in an unnecessarily obscure way and relies on unnecessarily rigid hypotheses.

Lyee is a tool that, when added to some language, helps programmers to perform their job. Moreover, Lyee is a tool that may become so easy to use that even non programmers can build up serious applications without ever trying to study any language. This is a substantial achievement and it is fair to fully acknowledge Mr. Negoro for such a relevant result.

Lyee is a tool for generating code in some language. The version we know generates code in Visual Basic, but in principle it can be tuned to any other language (be it VB, C++, Java, COBOL, or whatever any other language you may decide to work with). This implies that a proper interface between Lyee's tools and the supporting environment must be created.

The LyeeAll2 tool is constructed as follows: The project structure is expressed by tree structure. Upper frames shows descriptions of systems and defined and the lower frame [6] show the content of Process Route Diagram. When the user click on any of items, the icon

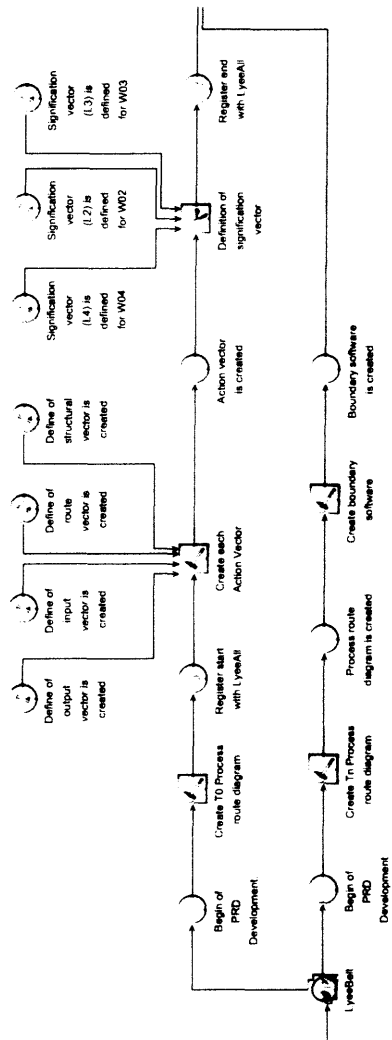


Figure 1: Lyees Development Process (Step 1/2)

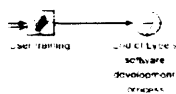


Figure 2: User verification phase and user training (Step 3)

becomes a blue sign implying that he is selected. The content of each window in the right work frame varies according to the selected item here. When the user specifies process route diagram in the upper frame, the details of the PRD is shown in the lower frame.

The Base Structure window shows us the structure of the Process Route Diagram. A Process Route Diagram specifies the navigation between the various components of the application. The process route diagram can be described as follows: The plural number of Scenario Functions are connected each other using the information provided by Route Vectors, and the connected Scenario Functions establish a structure that expresses an intent. The structure is called Synchronous Structure and the diagram of the connected Scenario Functions, is called Process Route Diagram, PRD. Process Route Diagram, on the other hand, can be drawn theoretically owing to the structural characteristics of SF and does not need to be defined in advance (see Figure 4).

The edit predicate window shows us the content of each box of signification vector. The seven kinds of rule of signification vector is:

1st Box: The 1st box of Predicate Structure is a rule to make judgment if the 2nd box of the same Signification Vector must be executed or not.

2nd Box: The rule of the 2nd box of Predicate Structure is to tentatively set a value to be set to the memory area of the 4th box.

3rd Box: The rule of the 3rd box of Predicate Structure is to implement significance judgment of the value obtained by the 2nd box rule.

4th Box: The rule of the 4th box of Predicate Structure is a rule to duplicate the value in the temporary area of the 2nd box into the actual area of the variable.

5th, 6th and 7th Box: The rules of the 5th, 6th and 7th boxes of Predicate Structure are not delivered axiomatically. They are prepared to match TDM to the mode of the current computer.

The List window shows us the content of the selected items. The registered content can be modified here. All the database operations are conducted on list window. With list window displayed at forefront in the right work frame, when items in the left tree view are clicked, the content of such items is shown. If it is not shown, it means that item is not used. Database operations are conducted on the list window displayed here. Files that the user needed for development work are stored in a work folder and the user can specify its path. This is not the folder in which LyeeALL2 is installed. There are a work folder to store project data and that to store system data. It is possible to specify separately or to specify the same folder.

For the agent the following attributes are mandatory:

- a) **User name:** For identification purposes the username of each agent should be stored in the database.
- b) **Password:** For identification purposes the password of each agent should be stored in the database.

For the athletes the following attributes are mandatory:

- a) **Club name:** For identification purposes the club name of each athlete should be stored in the database.
- b) **Birthday:** For identification purposes the birthday of each athlete should be stored in the database.
- c) **Name:** For identification purposes the name of each athlete should be stored in the database.
- d) **Address:** The athletes address is the data to be stored with each client/athlete. In real life there will be additional attributes as well. For studying purposes one attribute was sufficient.
- e) **Stage name:** For identification purposes the name of each athlete should be stored in the database.

A policy is characterized by the following attributes:

- a) **Start date:** The start date is the point of time when the period of validity of the policy starts.
- b) **End date:** The end date is the point of time when the period of validity of the policy ends.
- c) **Monthly rate:** The monthly rate is a number indicating the amount of money the athlete has to pay for the policy every month of the period of validity.
- d) **Policy type:** The type of any policy should be one out of legs, arms and head.

The AI Application System should provide the following functions:

- a) **Look details of my athletes:** This function should show a list with all athletes. For each athlete all policies held by the athlete should be displayed with their monthly rate and there stage name and club name.
- b) **Registry and add new policy for my athletes:** Using this function, a user should be able to add a new policy to a certain existing athlete.
- c) **Modify my athlete:** The data of a certain athlete should be modified using this function.

Project work folder consists of the following subfolders.

- a) **TPL** : Template Folder
- b) **PWK** : Project Working Folder

System work folder consists of the following subfolders.

- a) **PLT** : Word List Folder
- b) **SRC**: Generated Source File Folder
- c) **SWK**: Working Folder
- d) **SLF**: Self Description Folder
- e) **MET**: Meta-definition Folder

The folder where LyeeALL2 is installed consists of the following.

- a) **Bin**: All LyeeALL2 applications and help files are stored.
- b) **DBtbl**: Script files to create the database for LyeeALL2 are stored.

But before one begins with the development is it very important to create an alias name in the BDE administration [6]. The alias name is assigned to the database file by using this. If the user want to create an alias name he must do the following: He selects at first Create New from Object menu. Then, he selects Interbase in the dialog box and clicks then OK. Then assign a name as an alias name. When the focus is shifted to server name in the right window, appears. Click this and specify the database file. At last we must to save our steps. But in order to work with LyeeALL2, we needs to be connected to the database file first. Then we select our alias name from File menu. The user must select the alias to be used on LyeeALL2 from the list on the left and click then on the right symbol then it will be added to the list on the right. These aliases are used by LyeeALL2. After we press on OK we can show our alias name in Tree View on the left of the main window. Now the development of software applications can begin [6].

4 Athlete Insurance Lyee Application System

Most companies in business for profit have something to sell. In our case it is the policy. AI Lyee System is a full service athlete insurance firm application. In this section, we describe our application that used the Lyee concept. This application enables the insurance agent to manage his athletes and to sell to them policies. He has the possibility to register, modify, delete and inform himself about his athletes. On the other side the user has the possibility to inform himself about his policies. The gathered requirements for our application can be split up into two parts: requirements related to data representation and requirements for program functionality. Data Representation for major data entities were given: athletes, policies and insurance agents should be stored into one database each. In the following we describe the application system. But at first we present the requirements of our system.

The Main Menu screen is composed of four buttons, one of the button (Athlete Login) let the user read information about his policies. But before he can read his information (client name, monthly payments, policy type, monthly rate) (see **Figure 3**), he must give his user name and password. The password and the user name identifies the user in our application system on which the user attempts to log in. After he reads his information about his policies, he can use the logout (see **Figure 3**). The Information menu is the final menu on the right end of the menu bar. It contains menu entries that provide instruction or documentation on the application. When the user clicks on the information button, the application will display the text entered by the user in the display area of the main application dialog window. The information button gives the athletes and the insurance agents information about the version and the AI application system.

At next we describe the last button on this main menu (Agent Login). At first the agent must to click on the Agent Login button. This action opens the Login Screen. If the agent noticed his username and password in this screen, he see then his main menu. The agent main menu is composed of three buttons, one of the button let the agent registry and add a new policy for his new athlete. This screen is composed of twelve fields, eight fields let the agent specify the personal indicated of the athlete and four to let the agent specify the criteria of the policies and the start and end date of the policies. Further we see two buttons, one to validate the entry, and one to go back to the agent main menu. If the button (Look details of my athletes) is selected, all data about his athletes is listed. Each athlete is identified by his name, club name, stage name, policy type and monthly rate. If the button (Modify my athlete) is selected the application will display the athlete list screen. All data about his athletes is listed. When he want to modify one of his athletes he must to click on him. After he denoted the athlete he must to click on the modify button. The logout button let the agent go back to the initial window. The Screen Flow Chart shows us the relationship between all windows (see **Figure 3**).

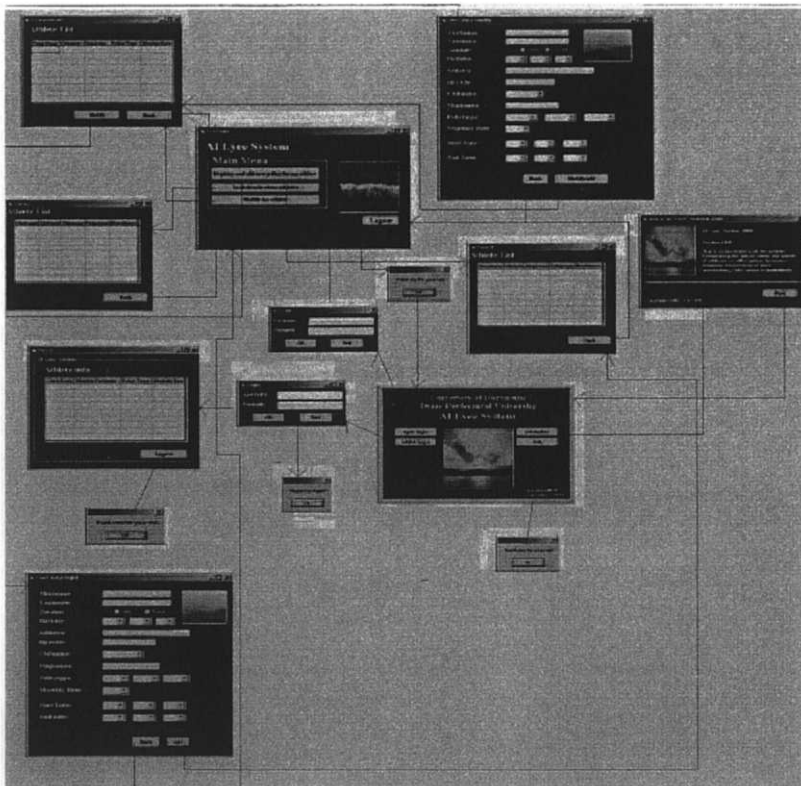


Figure 3: Screen Flow Chart

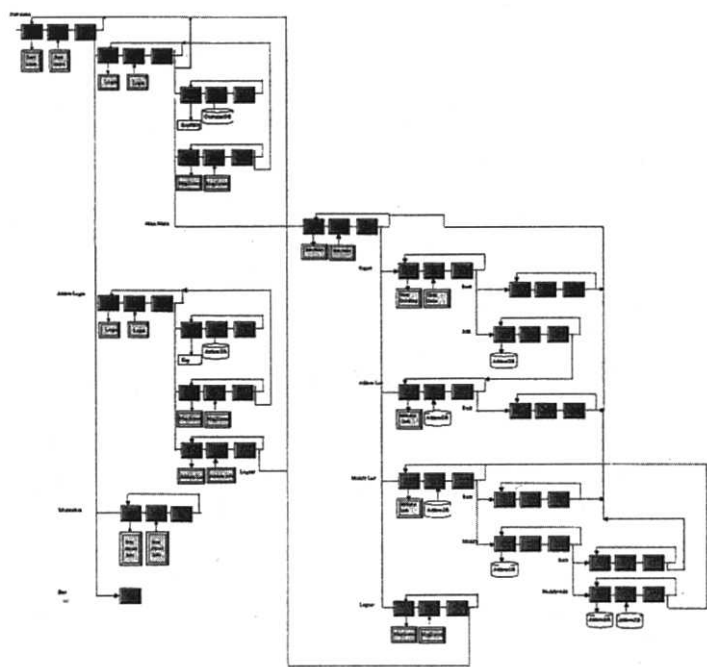


Figure 4: Part of Process Route Diagram of the AI System

5 Summary and Future Work

A lot of work has already been done so far. In particular we managed to be acquainted with Lyee theory and the LyeeAll2 tool, although the English version of the LyeeAll tool caused a lot of insuperable problems. With hard stepwise work it was nevertheless possible, to gain deep insight into the Lyee software development process, even though not all problems are solved yet. The software process described in this paper is based on a small example developed by one individual software developer. Although the programming techniques are the same as in big software projects, the largeness of projects has a significant impact on the process itself. In future it is especially interesting to see, how well Lyee software processes behave with strict boundary conditions according to the total time and costs of software development in comparison to object oriented and component based software development processes. The understanding of Lyee software processes is not only important to make the process comparable, it is also the basis of developing process enhancements and to evaluate, how far techniques from classical software development need to be extended to bring benefit to Lyee methodology. In our work we are especially interested in enhancing the Lyee software processes with configuration management techniques.

References

- [1] A. Oberweis : *An Integrated Approach for the Specification of Processes and Related Complex Structured Objects in Business Applications*, Decision Support Systems, 17:31-53, 1996
- [2] B. Berliner : *Parallelizing Software Development*, In Proceedings of the USENIX Winter 1990 Technical Conference, pages 341-352, Berkeley, CA, 1990
- [3] B. Westfechtel, B. P. Munch, and R. Conradi : *A layered architecture for uniform version management*, IEEE Transactions on Software Engineering, 27:1111-1133, December 2001
- [4] F. Negoro : *Principle of Lyee Software*, Proceedings of the International Conference on Information Society in the 21st Century, pp.441-446, Japan, November 2000
- [5] F. Negoro : *A Proposal for Requirement Engineering*, Vilnius Lithuani, Proceedings of the ADBIS, September 2001
- [6] R. Ijioui, R. Poli, H. Fujita, T. Kawakami : *The World of Lyee*, Part of the Lyee International Workshop, Lyee W02, Sorbonne Paris, Oktober 2002
- [7] V. Gruhn and R. Jegelka : *An Evaluation of FUNSOFT Nets*, In Second European Workshop of Software Process Technology, Trondheim Norway, September 1992
- [8] V. Gruhn : *Validation and Verification of Software Process Models*, PhD thesis, University of Dortmund, 1991
- [9] V. Gruhn, R. Jjioui, D. Peters, C. Schaefer : *Mid-Term Report for the Work Plan of the Collaborative Research Contract between Iwate University and University of Dortmund*, Lyee Intermediate Report, 2002
- [10] W. Deiters and V. Gruhn : *The FUNSOFT net approach to software process management*, International Journal of Software Engineering and Knowledge Engineering, pages 229-256, 1994

This page intentionally left blank

Invited Presentation 2

This page intentionally left blank

A User Centric View of Lyee Requirements

Colette ROLLAND
Université Paris I Panthéon Sorbonne
CRI, 90 Rue de Tolbiac
75013 Paris, France
Tel. 33 1 44 07 86 34 – 33 1 44 07 86 45
Fax. 33 1 44 07 89 54
rolland@univ-paris1.fr

Abstract. The paper deals with the modelling of Lyee user requirements and guidelines to support their capture. The Sorbonne contribution to the Lyee collaborative project aims to reduce the software development cycle to two explicit steps, requirements engineering and code generation by coupling the code generation features of LyeeALL with an interface to capture user requirements. The paper presents a 2-layer meta-model relating the set of concepts to capture user requirements to the set of concepts for the formulation of software requirements that are the input of the LyeeALL generation mechanism. It exemplifies the concepts with example and introduces the guidance support for capturing these user centric requirements.

1. Introduction

The research of the Sorbonne group within the Lyee¹ collaborative project is aimed at developing a methodology that supports software development in two steps, requirements engineering and code generation. The former is the contribution of the Sorbonne group whereas the latter is provided by LyeeALL.

LyeeALL is a commercial Japanese CASE environment which aims at transforming software requirements into code. As shown in Figure 1, the underlying Lyee approach [16] [17] comprises an *original framework* to structure programs, an *engine* to control their execution and a *generation mechanism* to generate programs from given requirements. These requirements are expressed in rather low-level terms such as screen layouts and database accesses. Moreover they are influenced by the LyeeALL internals such as the Lyee identification policy of program variables, the generated program structure and the Lyee program execution control mechanism. As a consequence it is difficult to get the Lyee customer away from the burden of Lyee internals instead of focusing his/her attention on the requirements. Projects conducted in industry with LyeeAll show the need to separate clearly software requirements from user-centric requirements in order to acquire the former from the latter.

¹ Lyee, which stands for Governmental Methodology for Software ProviDence, is a methodology for software development used for the implementation of business software applications. Lyee was invented by Fumio Negoro.

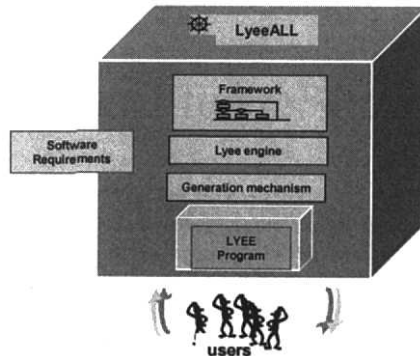


Figure 1 : LyeeALL

The Sorbonne group develops research towards meeting this need. As a first step, the group is aiming at:

- (1) defining a user-centric requirements model
- (2) developing methodological rules to support the capture of these requirements in a systematic way,.
- (3) developing a software assistant to guide the capture of user centric requirements
- (4) generating the Lyee software requirements from these user requirements

In a second step, the objective is to provide an intelligent software support for the elicitation of high level requirements and the automated generation of the Lyee software requirements.

In this paper we concentrate on points (1) and (2) above. In the next section we introduce the meta-modelling approach which was used to define the user-centric requirements model and we provide an overview of the model. Section 3 contains a description of the model concepts and illustrates them with examples. The next section deals with the process support to help in the capture of user-centric requirements. Some idea of future work is given in the conclusion.

2. Meta-Modelling Approach and Lyee Requirements Meta-Model

At the start of the project, it was quickly realised that Lyee was understood in operational terms such as Process Route Diagram (PRD), Pallets, Signification Vectors, Routing Vectors and the like and it was difficult to get a global, *systemic view* of it. The need for the latter was felt particularly strongly because :

- (a) user-centric requirements are to be related to Lyee software requirements. and a systemic model would help in clearly expressing this relationship.
- (b) additionally, the transition from user requirements to Lyee programs called for traversal across different levels of abstraction, a task that the area of modelling and meta-modelling is known to perform effectively.

Meta-modelling is known as a technique to capture knowledge about methods. It has been used for understanding, comparing and evaluating methods [7]. Meta-models were also used as a basis for method-engineering [4] and Case shell construction [3] [9] [13]. A number of meta modelling languages have been proposed to deal with (a) the representation of the product aspects of methods [1] [3] [8] [10] [15] [24] and (b) for modelling the process aspects of methods [12] [20] [21] [23].

We used a meta modelling approach to first model the set of concepts underlying the Lyee software requirements and secondly, to abstract from them the user-centric requirements model. The result of this effort is a 2-layer meta model² expressed with UML notations. The upper layer corresponds to the user-centric requirements model whereas the lower layer identifies the set of concepts required to express software requirements in Lyee terms.

Figure 2 shows the meta-model and highlights the separation between user requirements concepts and Lyee software requirements concepts. The former constitute the user requirement layer whereas the latter form the Lyee software requirements layer.

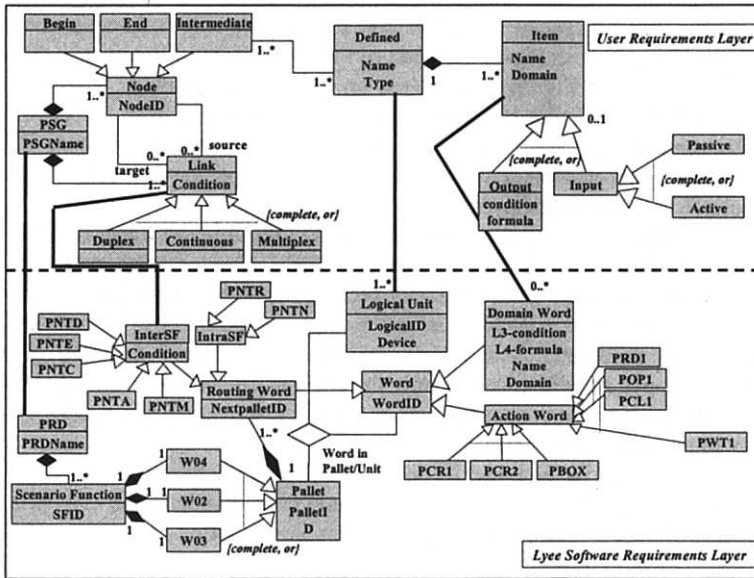


Figure 2 : Lyee meta-model

Let us introduce first, the lower layer concepts to express Lyee software requirements. The essence of the Lyee approach is to reduce software requirements to the description of *program variables* called *words*, and to generate the *control structure* that logically processes these variables and produces the expected result. Despite the traditional design approaches in which both the variables and the control structure of the program must be designed, LyeeALL generates the latter provided an appropriate description of the former is given.

- From the design view point, the approach can be compared to declarative approaches for information system design. In these approaches, system design is reduced to a set of predicates from which the state of the system can be derived at any point of time t . Lyee relies on the notion of *word* and makes the distinction between input words (these are given value through system communication with the external world) and output words produced by the system. Instead of predicates, Lyee uses formulae to express how to produce an output word. The ordering of word production does not need to be given. In this sense the Lyee approach is declarative.
- From the generation view point, LyeeALL is similar to a forward inference engine of an expert system which generates new facts by applying to the existing base of facts at time t those rules having their premises true. Similarly, the Lyee engine saturates the

² The term meta-model is used in the paper in the same sense as the term meta schema.

application of formulae till all the output words are determined. However, as the Lyee engine controls the execution of formulae which are procedural rules and not inference rules, the engine activates a proprietary function to a Lyee specific program structure, the *Process Route Diagram* (PRD). This structure is hierarchical : a PRD is composed of *Scenario Functions* (SF), composed of *Pallets* which are made of *Vectors*. In order to carry out the generated program control the function generates its own words, such as the action words related to vectors and routing words to distribute the control over the various SFs of a PRD.

The concept of *Word* is therefore central to the expression of Lyee software requirement, whereas the ones of *PRD*, *SF*, *Pallet* and *Vector* required by the word processing mechanism of LyeeALL are also part of the Lyee software requirements model. These concepts can be seen in Figure 2 as part of the lower layer of the meta-model.

The upper layer of Figure 2 is centred on three concepts only : *Defined*, *Item* and *PSG*. This reflects the fact that the user-centric model abstracts from the details of Lyee software requirements to identify the minimum set of concepts to capture the domain dependent requirements. However the simplicity of the upper layer results fundamentally from the declarative approach of Lyee.

We present the upper layer concepts in the next section and illustrate them with the *Split* example. *Split a Goal* is a functionality which, given a goal statement such as '*Withdraw cash from an ATM*', automatically decomposes it into a *verb* and its *parameters*. For example, *Withdraw* is the *verb*, *Cash* is the *target* parameter of the verb and *from an ATM* is the *means* parameter. The full functionality identifies 7 different parameters . However, in this paper we will consider only the two parameters exemplified above, *target* and *means*. Besides, the case considered in the following extends when necessary, the *Split* functionality in three different ways :

- (a) the storage of the goal and its decomposition in a database.
- (b) the retrieval of the goal name from a Goal table in a database.
- (c) the possible failure of the goal decomposition function.

3. The User Centric Requirements Meta-Model

- *Interaction driven user requirement capture*

In order to comply with the Lyee approach, the user requirements model should be centred on a notion which abstracts from the Lyee internal concept of word. Obviously words required by the Lyee processing mechanism are not relevant at this level. On the contrary, the concern is only with domain dependent words. Besides, there is a need to provide the requirement holder with a means to grasp a 'set of 'words' conceptually associated with one another. We propose the notion of '*system interaction*' for that purpose. We believe that the Lyee approach, which is output driven, fits with a use case [6] kind of user requirements capture.

Our suggestion to the Lyee user is to reason in terms of a *goal driven interaction* as shown in Figure 3. The interaction is meant to be between the user and the system viewed as a black box. The interaction is *goal driven* in the sense that the user asks the system to achieve the goal he/she has in mind without knowing how the system will do it. The user provides some input and receives the output which corresponds to the expected result. It is the achievement of the goal which produces the *output*. The *input* is necessary to achieving the goal. We refer to this goal as the *interaction goal*.

In generic terms, any interaction is characterised by the user goal '*Get a result*'; it produces an *output*, given some user *input*. In the *Split* example, the user goal is to get

support from the system to decompose a goal statement. Thus, 'Split a Goal' is the interaction goal. If, for example, the input is the goal statement 'Withdraw cash from the ATM', then the achievement of the goal produces the output i.e. the decomposed form of the goal : Withdraw verb cash target from the ATM means.

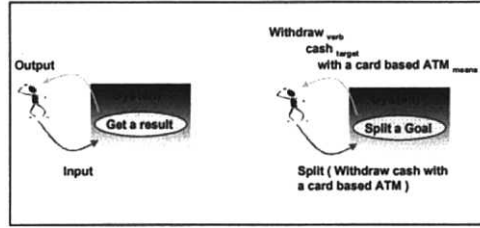


Figure 3 : The interaction view point

- *Words in interaction*

An interaction delineates a number of input and output 'words' logically assembled together. The former correspond to meta-model *items* belonging to the same *defined* (see below).

In order to systematise the collect of requirements, we identify generic classes of 'words' that will be instantiated in any such interaction and represented as items in the requirements formulation. Let us introduce so far three of them :

- W_{input} : the *input* provided by the user
- W_{result} : the *result* of the goal achievement
- W_{output} : the *output* displayed to the user

In the 'Split a Goal' interaction of Figure 4, W_{input} is the goal statement given as input by the user, the result W_{result} produced by the achievement of the goal 'Split a Goal', is the set $\{verb, target, means\}$ and the output W_{output} presented to the user is identical to the result, i.e. the set $\{verb, target, means\}$.

$$W_{input} : \{goal,\}$$

$$W_{result} : \{verb, target, means\}.$$

$$W_{output} = W_{result}$$

As illustrated with the Split example, the set of output words W_{output} might be the same as the set of result words W_{result} ; however the semantics is different as the former are the ones whose values are presented to the user whereas the latter are the ones resulting of the interaction goal achievement.

In addition, as shown underneath a relationship can be established between the W_{input} and W_{result} :

$$W_{result} \leftarrow W_{cmd}(W_{input})$$

Indeed, to get the interaction goal achieved, the user has to provide the input and to give some kind of command (W_{cmd})

- *The concepts of Defined³ and Item*

All the words of an interaction shall be represented with the meta-model concept of *Item*. The above typology helps identifying the items to be identified and described for a given interaction.

³ In the following concepts are in italics with a capital as first letter. Instances of concepts are in italics with a small first letter. For example, *Item* refers to a concept whereas *item* refers to an instance of the concept *Item*. A specific item such as *goal* is in small letter and italics.

$$Item \leftarrow W_{result}, W_{cmd}, W_{input}, W_{output}$$

The *items* belong to the same *defined*. A *Defined* is a container of *Items* logically related to each other. Defined and Items are the keys to expressing user requirements compliant to our meta-model.

In the Split example there are 5 *items*, namely *goal*, *cmdSplit*, *verb*, *target* and *means*. All belong to the same *defined*, *Split1*. Figure 4 presents the instantiation of the meta-model for formulating the Items and defined of the Split example. The instance is drawn with the UML object diagram notations.

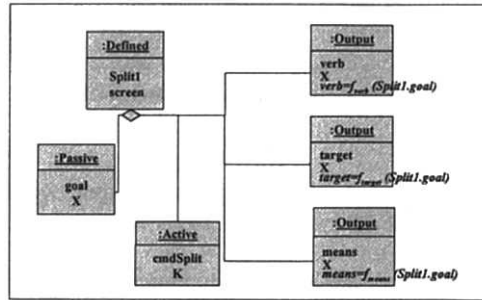


Figure 4 : Items & Defined of the Split interaction

In the meta-model, the concepts of *Defined* and *Item* have attributes: Every *defined* has a *name* (*Name*) and a *type* (*Type*) which identifies the physical device of the container (screen, file, database etc...). An *item* has a *name* (*Name*) and a *domain* (*Domain*): numeric (9), char(X) and (K) for screen buttons. In the Split example, *goal* is the name of an *item* of the *defined* *Split1* which has a string of characters as *domain*(X).

The meta-model specialises *Item* into *Output* and *Input*. An output is produced by the system whereas an input is captured from the user. *Input* is further specialised into *Passive* and *Active*. An active input triggers a system action whereas a passive input represents values captured from the user. A screen button such as *cmdSplit* in the Split example is an active item whereas *goal* is a passive one. Both specialisations (Item into Output or Input and Input as Active or Passive) are partitions of the set of *items* i.e. they are complete and exclusive.

Finally, the concept of *Output Item* has two specific attributes : *Formula* and *Condition*. The *Formula* is mandatory whereas the *Condition* is set by default to true. Due to the declarative nature of the Lvee approach, calculation dependencies among items do not need to be expressed through conditions. Therefore, only constraints such as validity constraints on input items might become conditions associated to outputs depending on the validity of these inputs. The formula is the calculation rule. In the Split example, the verb will be associated to the formula, $verb = f_{verb}(goal)$. The function f_{verb} when applied to a goal statement produces the verb of its goal statement.

- *Housekeeping goals*

The achievement of the interaction goal 'Get a result' is not always as straight forward as in the case considered so far. It can happen that it requires some additional goals to be fulfilled. We refer to these goals as *housekeeping goals*. Typical examples are the extension (a) and (b) of the Split case introduced above in the paper. In case (a) the decomposition is stored in the database and in case (b) the goal statement is retrieved from the database. 'Store Goal Decomposition' and 'Retrieve Goal Statement' are *housekeeping goals*. They are additional to the interaction goal 'Split a Goal'.

It shall be noticed that there is a fundamental difference between the two types of goals, *interaction goal* and *housekeeping goal*. Whereas the ‘*Get a result*’ type of goal is the essence of the interaction, the *housekeeping* goals contribute to the performance of a successful interaction but do not determine its purpose. The *interaction* goal is user-centric whereas the *housekeeping* goals are system-centric. Following goal decomposition in requirements engineering [3] [6] [23], housekeeping goals can be regarded as sub goals of the interaction goal ‘*Get a result*’.

However housekeeping goals implies new *items* and new *defineds* to be introduced. Let us understand the nature of these items by extending the typology of ‘words’ as defined previously in cases similar to extension (a) of the Split example. A similar reasoning can be done for each type of housekeeping goal [25].

In cases similar to extension (a) of the Split example, words to be memorised in a persistent manner such as in a database or a file, W_{indb} have to be identified as part of the user requirements formulation.

$$W_{indb} \leftarrow W_{cmddb}(W_{dbkey}, W_{output})$$

The above expression characterises the production of W_{indb} . In order to store output words W_{output} in specific database words, W_{indb} , the database key W_{dbkey} is required and the user shall activate a command, W_{cmddb} .

Consequently, new items shall be introduced :

$$Items \leftarrow W_{indb}, W_{cmddb}, W_{dbkey},$$

Housekeeping goals lead to specific *defineds* as they use a specific device distinct from the one characterising the defined of the interaction. In the Split example, the requirement formulation (Figure 5) includes the *defined GOAL* of type database with the associated *items*, *goalid*, *goal*, *verb*, *means* and *target*. In contrast, the W_{cmddb} is part of the defined associated to the interaction. In the Split example the command button, *CmdOK* is an *item* of the defined *Split1*.

As there are several *defineds* a precedence relationship between these shall be introduced. The concept of PSG in the meta-model captures this aspect.

- *The concept of PSG*

The meta-model includes the notion of a *PSG*, the *Precedence Succedence Graph* to stipulate ordering conditions between *Defineds*.

As shown in Figure 2, a *PSG* has *Nodes* and *Links* between *Nodes*. *Nodes* are classified into *Intermediate*, *Begin* and *End*. *Begin* and *End Nodes* are predefined nodes to start and end the program whereas *Intermediate Nodes* are related to *Defineds*.

Links between *Nodes* are of three different types : *Continuous*, *Duplex* and *Multiplex*. Whereas all links indicate the processing order of the related *defineds*, a *continuous link* is a forward link between two *defineds* while *duplex /multiplex links* are backward links between two *defineds*. The choice between a *duplex* or a *multiplex* link depends on whether or not data have to be transferred to process the backward *defined*. In the Split example, the *defined GOAL* is multiplex- linked to the *defined Split1* to get back to an empty Split screen after a goal decomposition was performed. In this case there is no data transfer associated to the backward link to *Split1* and therefore, the *GOAL-Split1* link in the *psgSplit* is a multiplex one. It shall be noticed that this information is user driven : it is a user decision to choose an iterative process allowing to capture a goal statement and ask for its decomposition several times.

Finally, the meta-model shows that a *Link* might have an associated *Condition* which constraints its occurrence.

Figure 5 presents the instantiation of the meta-model to formulating the Split a Goal requirements in case (a). The instance is drawn using the UML object diagram notations. It shows that there are two *defined*, (a) *Split1* of type screen, gathering the input and output *items* of the interaction and (b) *GOAL* of type database composed of the *items* representing the attributes of the relational table to store the goal decomposition.

Split1 comprises *active items* (*cmdSplit*, *cmdOK* and *cmdCancel*) whereas *GOAL* has only passive items. Some *items* in *Split1* are typed *input* (*cmdSplit*, *cmdOK*, *cmdCancel*, *goal*) whereas the others are *output items* (*verb*, *target*, *means*). All *items* in *GOAL* are typed *output* as they are produced by the program and stored in the database. Each of the *output items* in the *defined Split1* are associated with a *formula* that is its calculation rule. In compliance with the meta-model, the *output items* of the *defined GOAL* have *formulae* which are rules for expressing that the values of the attributes of the database table *GOAL* are the ones of corresponding *items* of the *defined Split1*.

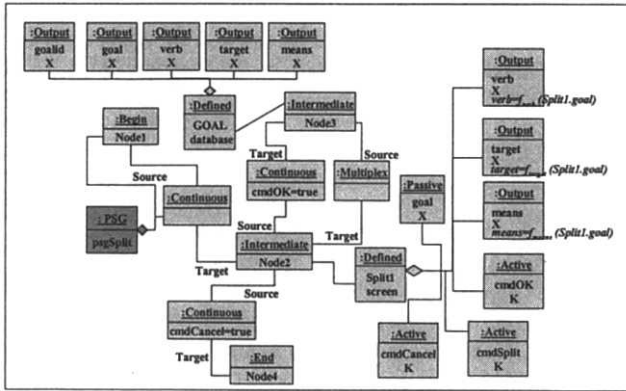


Figure 5 : Formulating the Split requirements through meta-model instantiation

The *psgSplit* comprises two *nodes*, *Split1* and *GOAL* in addition to the *Begin* and *End* nodes. They are related by a continuous forward link which is activated in the processing when the button OK has been pushed and a multiplex link in the backward direction which is processed as soon as the goal decomposition has been stored in the database.

- *Considering Obstacles to 'Get a result'*

The notion of *obstacle* has been introduced in requirements engineering by Colin Potts in [19] and further developed in [3] [29] [30] [31]. An obstacle is defined as anything which happens and causes a failure in achieving a goal. From a requirement viewpoint, it is important to identify obstacles as the system under construction shall be prepared to react to obstacle happenings. In our case, identifying the risks of interaction goal failure is a means to complete the requirements related to the interaction.

Considering obstacles to the achievement of 'Get a result' leads to the introduction of new types of words, W_{case}^i characterised as follows :

$$W_{case}^i = P(W_{output}) : f_{boolean} = true$$

$$W_{output} = \cup W_{case}^i$$

The set of words referred to as W_{case}^i corresponds to the subset of output words, W_{output} which are produced under a certain condition ($f_{\text{boolean}} = \text{true}$). The entire set of output words to be considered in the interaction is therefore the union of W_{case}^i .

Let us consider case (c) of the Split example, assuming that the f_{verb} function might fail if the name of the verb extracted from the goal statement is not in the table of verbs used by this function. The interaction might then, fail in achieving the interaction goal 'Split a Goal'. Consequently, there are two cases of output :

- case ¹ occurs when the verb, target and means items are presented to the user, whereas
- case ² occurs when the decomposition cannot be performed; the message 'Impossible Split' is shown to the user.

In this case $W_{\text{output}} = W_{\text{case}}^1 \cup W_{\text{case}}^2$
 $W_{\text{case}}^1 = \{\text{goal, target, means, verb}\}$
 $W_{\text{case}}^2 = \{\text{'Impossible Split'}\}$

An *item* has to be introduced for every word of each W_{case}^i .

Figure 6 shows the instantiation of the meta-model to formulating case (c) of the Split example. Two new *defineds* *Scase1* and *Scase2* have been added and linked to the *defined* Split1 through forward continuous links. These links are labelled with conditions identifying the two cases, case ¹ and case ². Each of the *defineds* aggregates the appropriate items : $\{\text{goal, target, means, verb, cmdOK}\}$ for the *defined* *Scase1* and $\{\text{M1, cmdOK}\}$ for the *defined* *Scase2*.

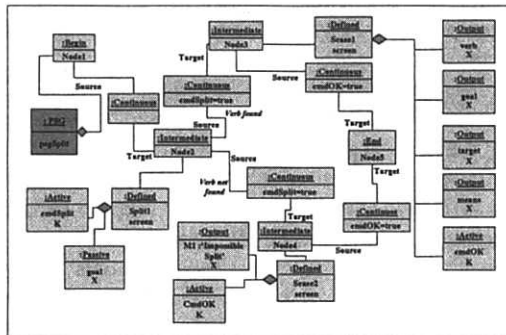


Figure 6 : Formulating the Split case (c) through meta-model instantiation.

• Compound interaction

In real projects the user has to deal with more gross-grained interactions than the Split interaction drawn in Figure 3. We suggest a distinction between a *simple interaction* and a *compound interaction*. The former is associated to one *single atomic interaction goal* whereas in the latter the goal is an *aggregate of interaction sub-goals*.

Figure 7 is an example of compound interaction where the goal 'Get Confirmed Booking' is an aggregate of two sub-goals : 'Request for Booking' and 'Confirm & Pay'. The request parameters are the inputs necessary to achieve the first sub goal which results in an offer to the customer. This offer is the input for the achievement of the second sub goal.

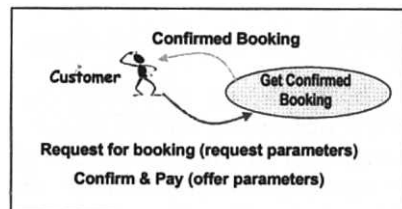


Figure 7: Booking interaction

The notion of AND decomposition of a goal is well known in requirements engineering [5] [6] [19] [22] and business process modelling [3] [11] [14] [18] [26] and seems to fit our needs. The interaction goal of a compound interaction is decomposable in two or more ANDed sub-goals. As shown in Figure 9, the interaction goal 'Get Confirmed Booking' is decomposable in two sub-goals 'Get Booking Offer' and 'Confirm Booking Offer'.

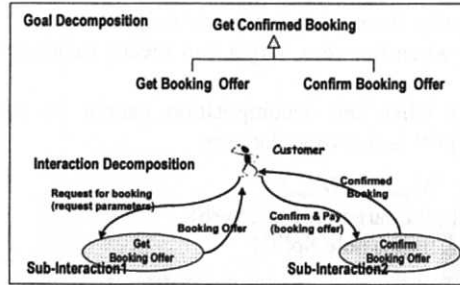


Figure 8 : Decomposition of the 'Get Confirmed Booking' interaction

It is important to notice that sub-goals are *interaction goals*. In other words, the compound interaction can be seen as a sequence of atomic interactions, each of them corresponding to one sub-goal of the compound interaction goal. This is exemplified in Figure 8 that shows the compound interaction to 'Get a Confirmed Booking' as composed of two interactions, the first one to 'Get Booking Offer' and the second one to 'Confirm Booking Offer'. Each of these interactions follows the pattern explained above and might include housekeeping goals. Each of these will have to be scrutinised as explained before to identify the involved *items* and *defineds*.

To sum up, the user centric layer of the meta-model identifies three key concepts, *Defined*, *Item* and *PSG*. These three concepts are used to express the set of domain dependent requirements and this expression is necessary and sufficient to derive the Lyee software requirements. *Items* are the essence of Lyee user requirements, the external form of Lyee internal words. A *Defined* is a group of items that are conceptually related to one another and are bound together in a simple or compound interaction. In addition to the *Defineds* flowing from the interaction, housekeeping goals introduce complementary *defineds* that require the use of devices such as databases, files or Internet communications. The concept of *PSG* captures the ordering of the *defineds* required by the user.

From a semantic viewpoint, this paper proposes to relate the user-centric requirements to the notion of an *interaction* and introduces an *interaction frame* with a *typology of 'words'* to reason systematically about the requirements implied by this interaction. It was shown that a complex interaction case can be mastered using a decomposition mechanism that breaks down the compound interaction in ANDed atomic interactions. This introduces the problem of guiding the process to capture user-centric requirements compliant with the meta-model. This problem is dealt with in the next section.

4. Guiding the Requirements Capture

Any method is defined as composed of a product model and a process model [20]. Whereas section 3 was dealing with the *product model* of the Lyee method, we consider here the *process aspect* of the method. Our aim is to systematise the capture of user-centric requirements and their formulation in terms which comply with the upper layer of the meta-model as presented in the previous section. Ultimately, our goal is to implement a software assistant to support the capture and formulation of these requirements.

Our process modelling approach is *Pattern* based. The concept of a pattern has been introduced by Alexander in architecture [2] and borrowed by IT engineers to capture software design knowledge. According to Alexander, a pattern refers to ‘a problem which occurs again and again in our environment and describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice’. The key idea of a pattern is thus, to associate a *problem* to its *solution* in a well identified *context*. The formulation of the problem and of its associated solution are generic.

We identified ten typical situations (*the problem*) in Lyee user-centric requirements capture (*the context*) and associate to them ten guidelines (*the solution*) to help in the requirements elicitation and formulation. We coupled the situation and associated guideline in a *Requirement Pattern* and therefore, the process model takes the form of a *Catalogue of Requirements Patterns*.

Each pattern captures a requirement situation and guides the formulation of the requirement in compliance with the requirement meta-model. In fact each pattern tells for the given situation, what are the concepts of the meta-model to instantiate and how, which are the attributes that have to be considered and what are the links between concepts that must be instantiated.

The ten patterns will be applied again and again in the different software projects using Lyee. Even if actual situations are different from one project to another, each of them should match one pattern situation and the pattern will bring the core solution to the requirements capture problem raised by this situation.

- *Identifying generic activities of requirements capture in an atomic interaction*

In order to systematise the requirements capture, we first founded our reasoning on the notion of atomic interaction and investigate the possibility to identify generic activities of requirements capture within the context of an atomic interaction. We end up with the view that the capture of requirements related to an atomic interaction comprises four activities to, respectively:

- Start the interaction (*To Start* requirement)
- Perform the action (*To Act* requirement)
- Prepare the output (*To Output* requirement)and,
- End the interaction (*To End* requirement)

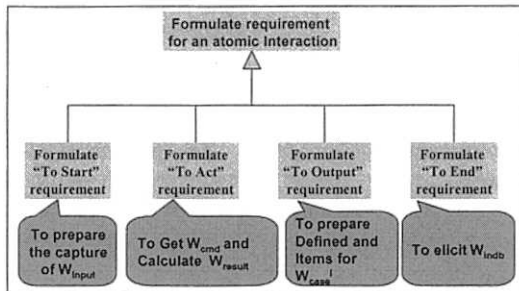


Figure 9 : Generic activities of requirements capture in an atomic interaction

As shown in Figure 9, each of these activities is linked to the ‘word’ typology introduced in section 3 as each activity is associated to one type of ‘words’. The requirement activity is concerned with the elicitation and definition of these ‘words’, their grouping in *defineds* and the positioning of those in the *PSG* of the interaction.

- The *To Start* requirement deals with the capture of W_{input}

- The *To Act* requirement is concerned by the elicitation of the W_{cmd} and the calculation of W_{result}
- The *To Output* requirement shall help eliciting and defining W_{case} ¹
- Finally, the *To End* requirement considers W_{indb}

- *Identifying typical situations in requirements capture*

The relationship between a requirement activity and its associated type of word was essential to identify generic situations for requirements capture. For instance, we identified two different situations dealing with the capture of W_{input} : either the input value is directly captured from the user or it is indirectly captured through the satisfaction of a housekeeping goal. In the Split example this corresponds to the initial case and case (b), respectively. In the initial case the user provides the goal statement whereas in case (b) it provides the *goalid* which is used to retrieve the goal in the database table.

We identified two generic situations for each of the four generic activities of requirement capture introduced above. These situations are described in the Table1 below.

Situation	Requirement Activity	Situation Characterization
S2	To Start	W_{input} are captured directly from the user
S3	To Start	W_{input} are captured indirectly through some housekeeping goal to retrieve the input value from a database or a file
S1	To Act	W_{result} are calculated by simple formulae which do not require the calculation of intermediate words
S8	To Act	W_{result} are calculated by complex formulae which do require the calculation of intermediate words and possibly the access to data in a file or database.
S6	To Output	There is no obstacle neither in the capture of W_{input} nor in the production of W_{result}
S7	To Output	A number of different cases of output production shall be considered due to possible obstacles either in the capture of W_{input} or in the production of W_{result}
S4	To End	The interaction ends normally without additional housekeeping activity.
S5	To End	Some housekeeping activity shall be performed such as storing part or the totality of $W_{outputs}$

Table 1 : Generic situations in requirements capture

It shall be noticed that the two situations of each activity are orthogonal. Given an interaction and one requirement activity , let say 'To Act' either S1 or S8 will be true but not both at the same time.

- *Identifying requirements patterns*

- *Atomic interaction patterns*

To each of the 8 situations of requirement capture presented above, we define a guideline that helps in the performance of the requirement activity. As the result of any of these requirements activities is an instantiation of the meta-model concepts, guidance tells which *items* shall be introduced, to which *defineds* they must be associated and how these *defineds* must be positioned in the PSG. Every guideline provides exactly this type of knowledge : given the situation at hand, the guideline advises on *items*, *defineds* and their *attributes* as well as *defineds precedence relationships* required by the situation.

We couple the *situation* and the *guideline* in a pattern, namely a *Requirement Pattern*. Figure 10 shows the 8 patterns corresponding to the 8 situations described in Table1. These are *atomic* patterns in the sense that they do not call for applying other patterns.

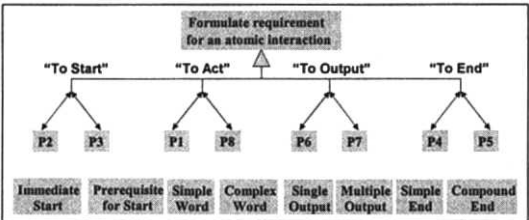


Figure 10 : Requirements Patterns for an atomic interaction

These 8 patterns provide advice to capture and formulate requirements for each of the generic requirements activities:

- P2 & P3 support the 'ToStart' requirements activity, i.e. the setting of requirements to ensure that Winput will be properly defined
- P1 & P8 help in the elicitation of requirements which guarantees that Wresult can be calculated by the Lye program
- P6 & P7 advice in discovering obstacles to interaction goal achievement and to formulate the appropriate *items*, *defineds* and *PSG links* for handling these obstacles in the Lye program.
- P4 & P5 ensure that the interaction will end correctly and that housekeeping goals will be taken care of.

- Composite pattern for atomic interaction

Each of the previous 8 patterns deals with one single requirement activity whereas to get the complete set of requirements for a given problem, the requirements engineer has to perform one of each type of activity. The complete set of requirements requires that each of the following be performed once: 'To start', 'To Act', 'To Output' and 'To End'.

To obtain advice on this, a new pattern, Pattern P9, is introduced. As shown in Figure 11, the requirement pattern P9 is a compound pattern composed of the 8 atomic patterns, P1 to P8.

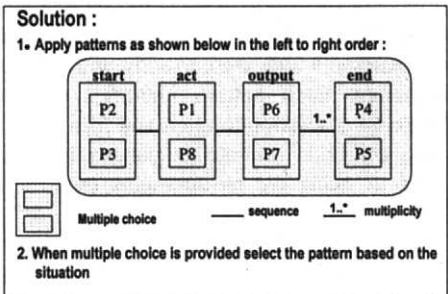


Figure 11 : The compound requirement pattern P9

P9 simply advises that one pattern for each of the four activities needs to be applied to complete one interaction requirements formulation. The choice of the right pattern to apply for each activity is based on the situation at hand. Since the situations of the two candidate patterns of any activity are orthogonal, the decision making is facilitated. For instance, in the simple case of the Split example (get the goal statement and outputs the

goal decomposition), P2 is applicable as the input is directly got from the user; P1 must be applied as the decomposition function produces the goal decomposition directly from the goal statement; P6 is the right pattern because there is no obstacle either in getting the input or in calculating the result and P4 is applicable in this case as there is no additional task to perform than displaying the goal decomposition to the user.

Thus, for a given interaction, the requirements process will consist of a path within P9. For instance, P2, P1, P6, P4 is the path for dealing with the basic Split example whereas P3, P8, P7, P5 is the path for the extended Split example (combining (a), (b) and (c)).

- Composite pattern for compound interaction

Finally, the requirement pattern P10 deals with a compound interaction as introduced in the previous section. As shown in Figure 12, P10 is a composite pattern which calls for the iterative application of P9.

As suggested by the figure, the pattern gives advice on how to decompose a compound interaction into atomic interactions to which the pattern P9 should be applied. In fact, the pattern helps in recognising that the interaction is not an atomic one in the first place.

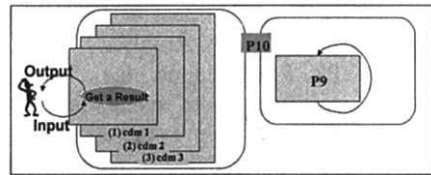


Figure 12 : The composite requirement pattern P10

5. Conclusion

The UP1 activity presented here relies on meta-modelling. Meta-modelling has been used in Information Systems as a way of developing abstractions of methods to aid in method understanding, evaluation and comparison. In extending this to Lyee we expected to gain a better understanding of how the Lyee method generates programs from given software requirements. Indeed the lower of the two layers of our meta-model achieved this purpose. The upper layer added a new abstraction level which makes it possible to deal with user requirements and not with low level software requirements. With this capability comes the possibility of generating Lyee programs directly from user requirements. The next step to be taken is to formalise the mapping rules between the two sets of concepts.

Meta-modelling addresses both, process and product aspects of methods. The meta-model presented in this paper is a product meta-model. To complete the formalisation of the method it is necessary to also model the way-of-working. The paper introduced the pattern approach and the ten patterns which are currently under development to support the acquisition of user requirements. Each pattern identifies a generic situation in user requirements capture and proposes a solution to elicit and formulate the requirement typical of this situation. The next step will be to validate the pattern through extensive experiments and to develop a CASE tool to guide the requirements engineers in the application of patterns.

6. References

- [1] N. Ahituv, 'A Meta-Model of information flow : a tool to support information theory', Communications of the ACM, 30(9), pp781-791, 1987.
- [2] C. Alexander, S. Ishikawa, M. Silverstein, M. Jacobson, I. Fiksdahl-King, S. Angel. 'A Pattern Language', Oxford University Press, New York, 1977.
- [3] A.I. Anton, W. M. Mc Cracken, C. Potts, 'Goal decomposition and scenario analysis in business process reengineering', Proceedings of the 6th International Conference CAiSE'94 on Advanced Information Systems Engineering, Utrecht, the Netherlands, Springer Verlag, pp. 94-104, 1994.

- [4] S. Brinkkemper, K. Lyytinen, R. Welke (eds): *Method Engineering : Principles of Method Construction and Tool Support*, Chapman & HALL, London, UK, 1996
- [5] J. Bubenko, C. Rolland, P. Loucopoulos, V. De Antonellis, *'Facilitating, 'Fuzzy to Formal' Requirements Modelling'*, Proc. of the First International Conference on Requirements Engineering, Colorado Springs, Colorado, 1994.
- [6] A. Cockburn, *'Structuring use cases with goals'*, 1995
<http://members.aol.com/acocburn/papers/usecases.htm>
- [7] Series of Proceedings of CAISE/IFIP8.1 International Workshop on Evaluation of Modeling Methods in Systems Analysis and Design (EMMSAD).
- [8] J.C. Grundy, J.R. Venable, *'Towards an integrated environment for method engineering'*, Proc. IFIP WG 8.1 Conf on 'Method Engineering', Chapman and Hall, pp 45-62, 1996.
- [9] F. Harmsen, S.Brinkkemper, *'Design and implementation of a method base management system for situational CASE environment'*, Proceedings of the 2nd APSEC Conference, IEEE Computer Society Press, PP 430-438, 1995.
- [10] A.H.M. Ter Hofstede, *'Information modelling in data intensive domains'*, Dissertation, University of Nijmegen, The Netherlands 1993.
- [11] R.L. Hsiao and R.J. Ormerod, *'A new perspective on the dynamics of information technology-enabled strategic change'*, Information Systems Journal, Blackwell Science, Vol. 8. No. 1, pp. 21-52, 1998.
- [12] M. Jarke, C. Rolland, A. Sutcliffe, R. Domges, *'The NATURE requirements Engineering'*. Shaker Verlag, Aachen 1999.
- [13] S. Kelly, K. Lyytinen, M. Rossi, *'MetaEdit+: A fully configurable, multi-user and multi tool CASE and CAME environment'*, Proc. CAiSE 96 Conference, Springer Verlag, 1996.
- [14] J. Lee, *'Goal-Based Process Analysis: A Method for Systematic Process Redesign'*, Proceedings of Conference on Organizational Computing Systems, , Milpitas, CA, pp. 196-201, 1993.
- [15] MOF Specification, OMG document ad/97-08-14, revised submission, September 1, 1997.
- [16] F. Negoro, *'Methodology to Determine Software in a Deterministic Manner'*, Proceeding of ICII, Beijing, China, 2001.
- [17] F. Negoro, *'A proposal for Requirement Engineering'*, Proceeding of ADBIS, Vilnius, Lithuania, 2001.
- [18] M.A. Ould *'Business Processes - Modelling and Analysis for Re-engineering and Improvement'*, John Wiley and Sons, Chichester, UK, 1995.
- [19] C. Potts, K. Takahashi, A.I. Anton, *'Inquiry-based requirements analysis'*, IEEE Software 11(2), pp. 21-32, 1994.
- [20] N. Prakash, *'On Method Statics and Dynamics'*, Information Systems, Vol 24, No 8, pp 613-637, 1999.
- [21] C. Rolland, C. Souveyet, M. Moreno, *'An Approach for Defining Ways-of-Working'*, Information Systems Journal, 1995.
- [22] C. Rolland, C. Ben Achour, C. Cauvet, J. Ralyté, A. Sutcliffe, N.A.M. Maiden, M. Jarke, P. Haumer, K. Pohl, Dubois, P. Heymans, *'A proposal for a scenario classification framework'*. Requirements Engineering Journal Vol 3, No1, 1998.
- [23] C. Rolland, N. Prakash, A. Benjamin : *'A Multi-Model View of Process Modelling'*, Requirements Engineering Journal (4)(4), pp169-187, 1999.
- [24] M. Saeki, K. Wen-yin, *'Specifying Software Specification and Design Methods'*, Proc. CAISE 94, LNCS 811, Springer Verlag, pp 353-366, Berlin, 1994.
- [25] TR5.1: *'L'Ecritoire Linguistic Approach : Concept Definition and Implementation'*. Technical Report, University Paris 1, C.R.I, mars 2002.
- [26] E. Yu, J. Mylopoulos, *'Using goals, rules and methods to support reasoning in business process reengineering'*. Proceedings of the 27th Hawaii International Conference System Sciences, Maui, Hawaii, January 4-7, Vol. IV pp. 234-243, 1994.
- [27] S. Si-Said, G. Grosz, C. Rolland, *'Mentor, A Computer Aided Requirements Engineering Environment'*, Proceedings of the 8th CAISE Conference. Challenges in Modern Information Systems, Heraklion, Crete, Greece, May 1996.
- [28] K.Smolander, K.Lyytinen, V.Tahvanainen, P.Martiin : *'Meta-Edit - A Flexible Graphical Environment for Methodology Modelling'*, Proceedings of the 3rd International Conference in Advanced Information Systems Engineering.
- [29] A. Dardenne, A.v. Lamsweerde, and S. Fickas , *'Goal-directed Requirements Acquisition'*, Science of Computer Programming, Vol. 20, 1993.
- [30] A.v. Lamsweerde, R. Dairmont, P. Massonet; *'Goal Directed Elaboration of Requirements for a Meeting Scheduler : Problems and Lessons Learnt'*, in Proceedings of Requirements Engineering, pp 194 -204,1995.
- [31] A.G. Sutcliffe, N.A.M. Maiden, S. Minocha, D. Manuel, *'Supporting Scenario-based Requirements Engineering'*, Transaction of Software Engineering, Special Issue on Scenario Management, Vol. 24, No. 12, 1998.

This page intentionally left blank

Chapter 4

Requirement Engineering and Meta Models

This page intentionally left blank

VOLYNE: Viewpoint Oriented Engineering of the Requirement Elicitation Process for Lyee Methodology

Pierre-Jean CHARREL, Laurent PERRUSSEL, Christophe SIBERTIN-BLANC
Université Toulouse 1 & Institut de Recherche en Informatique de Toulouse,
21 Allée de Brienne, F-31042 Toulouse Cedex, France.
Tel: (33) 561 12 87 99. Fax: (33) 561 12 88 80

Abstract. Lyee methodology regards the requirement process of new software from the principle that each actor of the designing project expresses intentions on the software to be designed. To improve the requirement elicitation process, actors are here considered from the so-called Viewpoint Paradigm, which considers the conditions for an object to be designed to acquire sense from multiple sources. The base of this paradigm is stated as follows: the sense of an object is the integration of the viewpoints which are exerted on it. The two concepts of Viewpoint and of Viewpoint Correlation are considered. Two Correlations are presented. The first aims at recognizing Viewpoints and their Correlations starting from the written documents produced during the requirement process. The second aims at managing the inconsistencies which occur during the process. The implementation of these principles in LyeeAll tool is finally addressed.

1. Introduction

The Lyee methodology is a method for software generation, based on the principle that the software reflects the intentions of the intentions of all the customers and users of the future product. Practically, the present CASE tool LyeeAll is able to automatically generate programs from a set of specifications [12]. These specifications are issued from a requirements elicitation stage and they are mainly described in terms of variables (words), relations between variables (functions), origins (screens, database).

During the early designing phases, i.e. the phases during which requirements are elicited and analyzed, the software exists only from the intentions of all the actors involved, i.e. the client, the contractor, the members of the project team and future users. To design the software, these Actors will be called upon to come up with a series of intermediate, transitory and accessory objects as well as specifications, prototypes, test sets, beta versions, etc., which can be seen as traces of the different viewpoints exerting an influence on the future software. These objects appear during the validation phases of the project when different viewpoints confront each other, correlate, and at last converge towards the final valid set of specifications. They are a series of representations of the future software. In other words, they are the traces of the constitution of the *signification* of the future software.

The aim of a requirement engineering activity is to define what the future software must do. So the goal is to obtain a complete and consistent set of the future software characteristics [18]. Multi-viewpoints Requirements Engineering has been aiming since the

early 90s' at handling in a better way a software specification and thus improving the quality of the output produced by the requirements elicitation and modeling stage. At the end of the requirements process, the resulting specification (of the services that has to be provided by the future software) has to be consistent and complete. In a distributed specification activity, each actor produces a specification fragment of the future software [18]. Each fragment must have the ability to fit in with the others so as to produce at the end of the process a complete and consistent set of specifications. This managing principle spares the obligation to seek global coherence permanently but requires any inconsistency. This management implies to tolerate inconsistencies during the process. Global consistency is only needed at the end of the process. Nothing dictates that the two tasks should be conducted simultaneously [3], [6].

In a multi-viewpoint requirements activity, each actor or group of actors produces a specification fragment. I. Sommerville and P. Sawyer [20] propose to distinguish two kinds of Viewpoints:

1. Viewpoints associated with the stakeholders;
2. Viewpoints associated with organizational approach: they usually bring constraints on the future software (cost, safety...).

The first kind of Viewpoints is close to the principle of Lyee since it directly addresses the entries of each individual actor of a designing project. Thus we focus on this first case where Viewpoints are exerted by different stakeholders. The requirements engineer will elicit the data and their associated processes with them. Usually a Viewpoint maintains some high levels goals describing his needs. Goals help to scope the Viewpoint [21] (cf. Fig. 1).

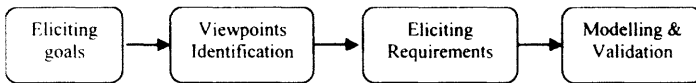


Fig. 1. The Multi-viewpoint Requirement Process

Section 2 presents the principles of the *Viewpoint Paradigm* and reconciles the precepts of semiotics on the conditions necessary for an Actor to give an Object sense. Section 3 defines the two key elements issued from the Viewpoint Paradigm when used in a Requirement Process: the two concepts of *Viewpoint* and *Viewpoints Correlation*. Section 4 presents three models of Viewpoint and Viewpoints Correlation closely related to Lyee methodology: sub-section 4.1 describes an experimental framework aiming at eliciting viewpoints and their relationships on the basis of written documents produced during the requirement collecting stage. It gives rise to operational concerns related to the intelligibility and control of the requirement process; sub-section 4.2 presents a model of Correlation which supports the definition of convergence indicators of the requirement process by means of a logical framework where each Viewpoint is associated with a knowledge-based system; sub-section 4.3 addresses the question of consistency management during the requirement process by means of a logical-based tool. Section 5 concludes the paper.

2. Viewpoints as a Paradigm to Design Requirements: a Semiotic Foundation

The sense of a paradigm given by Thomas Kuhn and whose characteristics are defined by Edgar Morin in [13] is the following: a paradigm is a vision of the world that is neither verifiable nor disprovable, which is accepted as an axiom, excludes the issues it does not

recognize, generates a feeling of reality and is recursively connected to the reasoning and systems it generates. Viewpoints, as a social issue, could be the key of such a paradigm. The statement on which this so-called *Viewpoint Paradigm* is founded is the following:

The sense of an object to be is the integration of the viewpoints exerted on it.

Two key points give rise to a definition of the elements in this paradigm: the viewpoint concept is central to two processes, the process whereby actors in the requirement process aim at designing software communicate amongst each other and the process whereby software, as an object, achieves sense.

The first key point arises out of the following observation: the act of software designing brings into play a great many technical, organizational and financial skills to find solutions to problems such as technical constraints, controlling cost prices, managing and coordinating teams, over a period that may last for a very long time. It can be said that all actors contributing one of these skills to the project has his/her "own" software that must be integrated with that of his/her partners.

The quality of communications between project participants is therefore a key factor to the success of the requirement process. Indeed, in this concurrent engineering activity, the actors are exchanging partial, incomplete and even contradictory representations. The basic idea is to not only take into account the representations of the future software, but also the software itself and its requirement process.

The second key point is to take into consideration the sense of objects and the actors that give these objects sense. In this way, the object to be designed and an actor participating in the project are not isolated entities: the object has sense when it is connected to how it is interpreted by an actor in a context through a representation that takes on the form of a statement using a symbology. Any representation of an object is thus subjective and contextual.

This position is conducive to a global – systemic – view of Objects, Actors, Representations and the requirement process: it identifies in a wider sense the object's sense and the result of the process used to design this object.

Viewpoints and Communication Process. The first steps in the software requirement process generally involve producing documents in natural language. "Formal" specification documents are only produced at the end of the process.

We encounter the first semiotic foundation of the Viewpoint concept, in connection with the French approach of semiotics.

According to this approach [1], [8], semiotics provides a tool for visiting a document like a monument. It thus presents text as being inseparable from the author and the reader: the sense of a text is that given to it by its author, which is also the sense retrieved by the reader. Thus we may observe that a text is a communication medium if a reader is able to retrieve a sense from it, albeit a different one. A text thus only contains conditions placed by the author for retrieving the sense, these conditions being relative to the form – structure, presentation –, the literary genre, the language – English, terminology –, the style, etc. A co-authored text is the product of a pooling of several sense retrieval conditions, i.e. those recognized by its authors.

The French semiotic method is conducive to highlighting the differences in a text. These differences are considered as the only sources of sense. They are elicited using three types of analysis. The first analysis consists of identifying "narrative programs" or contracts: these feature an "actant" – person or thing – that performs or has an action performed which is then evaluated. Its opposite is an anti-narrative program which creates an opposition and is a source of sense. The second analysis looks at how the roles and

interaction of the actants combine. Roles are expressed on the basis of canonical types. The third analysis identifies the "isotopies" of the text, i.e. its themes and the related elements of the text.

All these constructions are representations of the text. They throw direct light on invisible aspects of it. These are, in fact, relationships that can be presented in the form of graphs showing terms and concepts that although present in the text, cannot be isolated from it because of the latter's linear form.

Elements linked to the Viewpoint concept are thus found in these French semiotic analysis schemas: actors, objects, and relations between actors and objects.

Viewpoints and Meaning Process. Peirce's semiotics defines a sign as "something that takes the place of something for someone with some respect for some reason" [16]. According to Umberto Eco [4] "*With some respect*" means "that the sign does not represent the entirety of the object but rather – through various abstractions – represents it from a certain viewpoint or with a view to a certain practical use". Thus, according to Peirce, "nothing is a sign if it is not interpreted as such", and a sign only acquires the status of a representation of an object in a relationship of three terms encompassing the object, the sign as a signifier or expression, and the signified as the content of the expression.

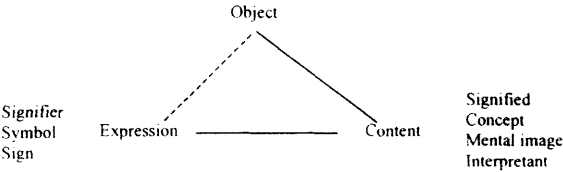


Fig. 2. The semiotic triangle

The triad < object, signifier, signified > is often represented in the "semiotic triangle". Fig. 2 shows this triangle and gives some of the equivalent terms used by Saussure, Morris, and Hjelmslev [4].

Peirce adds two notions to his semiotic triangle: the first is that a content can itself be an expression (a sign), and the second is that the context decides which content to associate with the expression to give it sense.

In the archetype of the signification process represented by the semiotic triangle we find several elements connected with our Viewpoint notion: the Object and the conditions for an Expression to be qualified for representing this Object.

3. Viewpoint and Viewpoints Correlation in a Requirement Process: Semi-formal Definitions

Reconciling the Viewpoint notion with semiotics first leads to a set based representation of the basic concept of Viewpoint and then to this of Viewpoint Correlation. Let us assume a given universe of discourse.

Viewpoint. A *Viewpoint* implements the conditions for an *Actor* to interpret the sense of an *Object*: it is defined by the *Object* on which the interpretation is performed, the *Actor* performing it, the *Expression* and *Content* of the interpretation of the *Object* by the *Actor*, and the *Context* in which this interpretation is performed.

A *Viewpoint* thus comprises five poles: the *Actor* holds at least one *Viewpoint*, in the *Context* of which he or she produces an interpretation of the *Object* to be: the *Object* is

interpreted by the Actors exerting a Viewpoint on it; the Context is the condition governing the way the Actor exerts his or her Viewpoint (the place from which the Viewpoint is exerted, the moment in time it is exerted, the tool used by the Actor to exert his or her Viewpoint...); the Expression is a statement, expressed in a symbolic system, that is attached to the Object by the Actor within the Context of the Viewpoint to express his or her interpretation of the Object; the Content is the sense given within the Context by the Actor to the Object by means of Expression.

The semiotic triangle does neither mention the Context in which the sign is produced nor the Actor that produces it. A Viewpoint can be considered to be a semiotic triangle situated for its Actor in his or her interpretation Context (cf. Fig. 3).

The Content, certainly the most "abstract" of the five poles, contributes knowledge given by the Viewpoint on the Object.

When the symbolic system used in Expression is a formal one, i.e. when semantics is associated with each statement, the two poles Expression and Content merge and correspond to what we already termed *Representation* at an earlier stage.

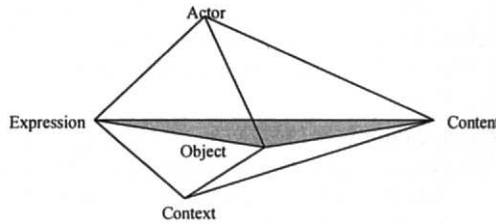


Fig. 3. Viewpoint and semiotic triangle

Universe of Viewpoints. The *Universe P* of viewpoints is the Cartesian product:

$$P = A \times O \times C \times E \times CO$$

in which A, O, C, E, CO designate respectively the aggregate of Actors, Objects, Contexts, Expressions, and Contents, each one being referred as the poles of the Universe.

We use dotted notation to designate one of the components of a Viewpoint. For example, $p.a$ designates Actor a of Viewpoint p in universe P .

For the following definitions, the Universe is implicit. Actually, it is the reference from which all the Viewpoints can be defined.

<X>-Correlation. We use *<X>-Correlation* to designate any transitive relationship on $X \times X$, where X designates one of the five poles

This notion is extended to Viewpoints in the following way: two Viewpoints p and p' are said to be *<X>-correlated* if an *<X>-Correlation* exists between two of their corresponding poles.

<X>-Correlator. An *<X>-Correlator* is a function:

$$cr: X \rightarrow X$$

where X is one of the poles.

Remark. The transitive closure of each $\langle X \rangle$ -Correlator cr defines an $\langle X \rangle$ -Correlation: $\{(x, cr(x)), x \in X\}$

This notion is extended to Viewpoints in the same way as $\langle X \rangle$ -Correlation.

Examples. In the universe of an information system to be computerized, the relationship "works in the same department" is an $\langle A \rangle$ -Correlation; "is the hierarchical superior of" is an $\langle A \rangle$ -Correlator over all Viewpoints; if the Expressions of Viewpoints p and p' are two knowledge bases made of logical formulae, an $\langle E \rangle$ -Correlator can define the formulae of p' deducible from the formulae of p .

System of Viewpoints. A System of Viewpoints is defined as the couple:

$$S = \langle P, CR \rangle$$

where P is a universe of Viewpoints and CR is a set of $\langle X \rangle$ -Correlators defined on P .

Graph of Viewpoints. A System S of Viewpoints is called a Graph of Viewpoints iff each Viewpoint in S is correlated to another through an $\langle X \rangle$ -Correlator of CR :

$$\forall p, p' \in P, \exists cr \in CR, (p, p') \in cr \text{ or } (p', p) \in cr$$

A Requirement Process as a Graph of Viewpoints. According to the Viewpoint Paradigm, the process whereby requirements are collectively designed gives rise to a particular subset of correlated Viewpoints which can be represented by a graph. Here, the Contexts of all Viewpoints are the various milestones of the requirement process. The first and last nodes of the Graph respectively relate to two Viewpoints whose Actor is the project's customer, and the intermediate nodes are the various Viewpoints exerted throughout the requirement process.

For the initial Viewpoint of the graph, the Object is the assignment on the Actor-customer's purchase order, the Context is the instant the project is launched, and the Expression is the entire set of documents produced by the Actor-customer to the Requirements team.

For the final Viewpoint of the graph, the Object is the produced set of requirements – and all knowledge acquired on its maintenance and operation –, the Expression is the integration of all the Expressions of the Viewpoints in the final Context, and the Content then represents the formal acceptance by the customer of the Object.

The Object only exists at the end of the requirement process, and it is the Object of all the Viewpoints exerted in the final Context of the process. The process is complete when the final Viewpoint of the customer Actor is able to prevail. At last, the Object acquired its sense for all the Actors involved in the requirement process, who exerted a Viewpoint on it.

Dynamic and Static Correlators in a Requirement Process. We classify the correlators into two types: Dynamic Correlators and Static Correlators.

Dynamic correlators are relative to the requirement process and support the intelligibility of the process: they ensure its visibility and consistency. The Context pole represents the time scale of the process. So, dynamic correlators are $\langle C \rangle$ -correlators, where Context is Time.

Static correlators are all $\langle X \rangle$ -correlators defined at the other poles. They can take on the form of reasoning on Viewpoints to organize, when necessary, their consistency.

Reasoning on Viewpoints and the process facilitates the discovery and management of all significant differences.

Examples. Two interesting situations can arise in the course of a requirement process, which are interpreted using Static and Dynamic correlators:

- “Validate a Representation” is a Static <R>-correlator which links the Viewpoint p1 of the Actor who produces the Representation to be validated and the Viewpoint p2 of the Actor who validates it. The <E> component is the identity relationship, and the <CO> component is the boolean function: $p1.CO \rightarrow \{true, false\}$.
- Objects are created throughout the process. They are assembled and modified in order to constitute the final body of specifications. Let us consider the history of the different versions of an Object produced by the same Actor during the process. Each version is related to one Viewpoint, and all these Viewpoints are linked to each other by <A, O, C>-Correlators where the <A> and <O> component are the identity relationship, and the <C> component is the relationship “next step”.

4. The Viewpoint Paradigm for the Intelligibility of a Lyee Requirement Process

The Viewpoint Paradigm generates specific representation models. Two models are presented in this section, for managing knowledge generated by Viewpoints exerted in a requirement process. The first model is particularly adapted to Lyee methodology because it aims at improving the elicitation of the “words” by the production of significant lexical networks from which one can extract the “words”. The second model uses a logical framework to address the management of consistency between requirements.

4.1 Infometrical Correlation

The first Correlation aims at gathering the so-called Defined and Items of Lyee from the documents produced during the requirement process.

Let us summarize a research experience the overall objective of which was to demonstrate that the notions of Viewpoint and Viewpoint Correlation could be used and useful to structure, understand and process the results of the content analysis of a discourse. The discourse was a transcript of a conversation between three space technology engineers, and the framework was a project for designing the architecture for a new small sized spacecraft. The objective was to provide tools to elicit the Viewpoints brought into play during the requirement process and to highlight those which have effectively contributed to the success of the process.

A proposition for a specification was drawn up during a meeting between the project manager, named D1, and two engineers, named D2 and D3. The meeting lasted three hours. An audio-video recording and the written transcription of the discussions held during the meeting constitute the records of the process.

Several analyses were carried out on the discussion transcript. One of them, carried out by C. Vogel [7]. This analysis used a statistical representation of the text as the source and not the text itself by means of the SEMIOLEX™ tool. The text was divided into six periods of thirty minutes. A glossary of keywords was selected from the general index of keywords constructed from the transcript. The keywords in the glossary occur at least twice in the index. On the basis of this glossary, the statistical analysis combines the two data

representation planes built from the various samples: frequency distribution tables and co-occurrence tables of the keywords.

Part of the results of statistical analysis is represented graphically by lexical networks attached to the co-occurrences. Fig. 4 illustrates an example of a network and the excerpts from the relevant text: the term "shielding" appears as a *pivot* for neighbouring terms such as "impact" or "pressure" and so defines a kind of proximity between them.

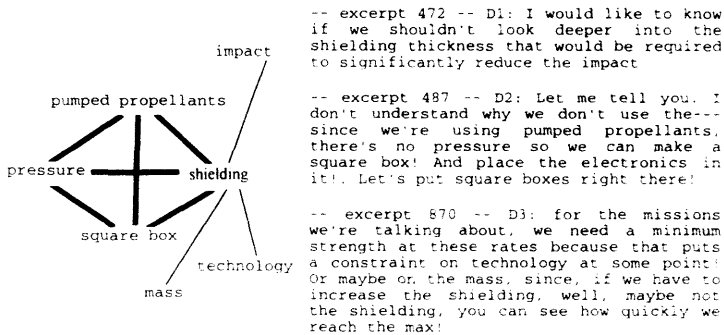


Fig. 4. Text excerpts with the lexical network which represent them

Let us formulate this situation in the Viewpoint Paradigm. The conversation transcript is the trace of the interventions of the Actors D1, D2 and D3, but also of other Actors mentioned by D1, D2 and D3 – individuals, companies –, of Objects produced by the latter – “shielding”, “pumped propellant” –, and of their interactions during the process with respect to the components of the Object being designed – layout of the satellite’s propellant tanks, size of the antennae, mass of the components, etc. This transcription can be interpreted as the trace of the requirement process of this Object and its operating environment.

The analyst of the situation is himself or herself an Actor who exerts a Viewpoint: the Object of which is the requirement process and its related graph of Viewpoints, the Context is the period of time the meeting lasted between the initial instant at which the Object was non-existent and the final instant at which the conversation was stopped by the project manager D1, and the Representation is the transcription of the discussion. This Representation is also that of the Viewpoints exerted by the three Actors in the meeting on the Object to be designed. The Representations of these Viewpoints are the transcribed interventions of the Actors during the meeting, and their Contexts are the instants at which these interventions took place. The Object is constructed as dialogue progresses by adding and updating Representations in the Viewpoints brought into play.

This experiment led to define a general dynamic Correlation, the *Infometrical Correlation* based on lexical networks. The infometrical aspect of the Viewpoints enriches the analysis, by offering selected views of the corpus owing to the networks of co-occurrences. Among these views are: the network local to a Viewpoint, by selecting terms which co-occur with the Actor of the Viewpoint; the network limited to a given instant of the designing project; the networks whose forms stand out.

This experimental framework can be applied to Lyee methodology in the following way: the notion of Term has to be replaced by the Lyee characters, i.e. Words, Defined, Items.

Conditions, Formulas such that Lyee engineer is able to recognize easily in the final lexical networks an initial classification.

For example, a term such *customer ID* is expected to meet the other terms *screen*, *database*, *input*, *output* in lexical networks such as his of Fig. 5. They are to be recognized in the transcriptions of interviews of the Actors-customers and Actors-users and so improve the capture of the corresponding entries of LyeeAll.

Another possibility is to implement the following Object model of the Lyee structure in the model of Fig. 5. The Term class is replaced with the Word class and the following class diagram.

Fig. 5 presents a class schema in the UML notation [19] which suits as the Infometrical Correlation of the Viewpoints. The corpuses of written documents, from which terms are issued, are accessible via the "source documents" method of class Representation. The "classification" method of class Viewpoint represents the access to classification tools as used in the experiment.

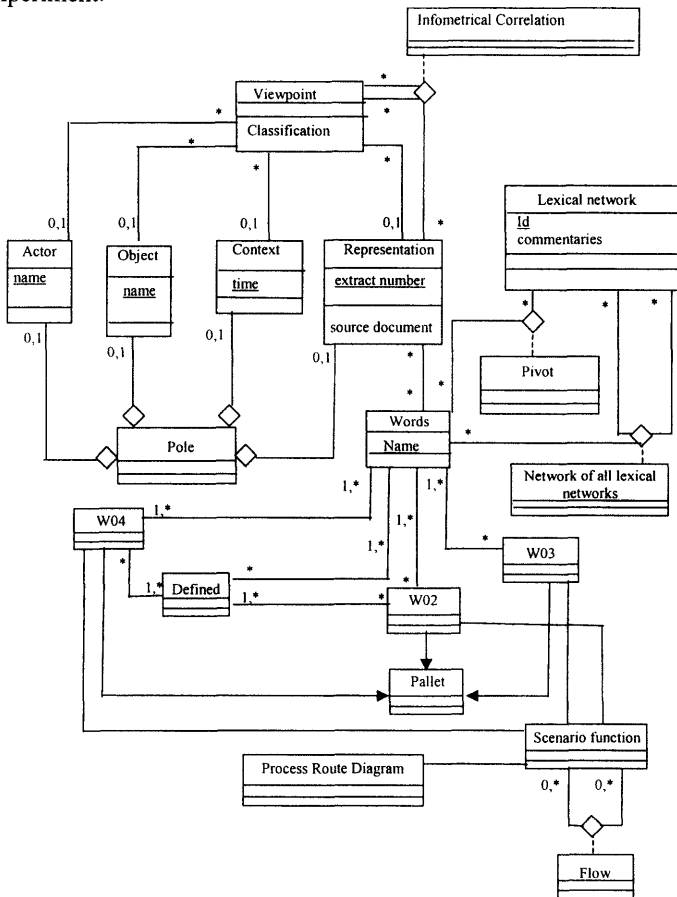


Fig. 5. Object-oriented model of an Infometrical Correlator

The Infometrical <R>-Correlator gives access to several analyses, accessible by a query language or a visualization tool: the number of links between a term and other terms or other networks; the integration or isolation of a network – number of adjacent networks –,

the density of a network – number of linked terms, number of links in the network; the dimension of the network constituted by all the networks – number of nodes, number of links; the network density – maximal distance between two nodes, ratio number of networks / number of terms, ratio number of networks / number of links.

For corpus of texts where chronology is a significant parameter – as the documents of a project – a computer-aided management tool of the history of the Representations of the Viewpoints can provide qualitative measures like: the trajectory of a term in the different Viewpoints, i.e. its appearance and disappearance, increase of co-occurrences with other terms or progressive isolation; scenarios issued from an infometrical analysis, i.e. recognition of regularities in the networks with reference to types of scenarios, acquisition of new scenarios.

These lexical networks allowed terms of the discourse appear brightly to the requirement engineer and the three designers across of the experiment.

4.2 Management of Consistency between Requirements

The Viewpoint Paradigm get rid us of the constraint of a permanent and total consistency between the different Representations. This tolerance for inconsistencies allows to not restraining prematurely the requirement process, and, for example, allows a better investigation of alternatives. In contrast with a centralized process whose goal is to prevent any inconsistency in the requirements, the goal in a distributed activity is to manage the inconsistencies. These characteristics are also found in the ideas put forward by S. Easterbrook et al. [3], [2] or by B. Nuseibeh et al. [6], [15]. Notably they claim that removing the perfect consistency constraint allows a better concentration on the requirement activity. Thus, the requirement expression appears less important than the expression of relationships between different Representations, i.e. the <R>-Correlations of the Viewpoint Paradigm. Here, we focus on the expression of particular <R>-Correlations.

At some instants of the Requirement Process, the Actors must confront part of their produced requirements with those of the other Viewpoints. In concrete terms, in each Representation, there are some elements which allow establishing an <R>-Correlator cr with some elements of the other Representations:

$$\forall p1, \exists p2, p1.r \neq p2.r \text{ and } (p1, p2) \in cr$$

A similar formulation of these <R>-Correlations is also found in [15]. Now, we have to deal with the different kinds of <R>-Correlations. For instance, let us consider semantic links where Actors agree (or seem to agree) on a terminology.

Let $p1.R$ and $p2.r$ the respective Representations of two Viewpoints $p1$ and $p2$ of P . Each Representation is a set of items (e.g. the statement of an atomic requirement expressed as a logical formula). The consistency of the Viewpoints $p1$ and $p2$ can be evaluated by means of <R>-Correlations like the followings six ones [17], where e is an element of $p1.r$ and e' is an element of $p2.r$ deduced from e by means of some transformation:

- $(p1, p2) \in R1$ iff $p1.r \neq \emptyset \Rightarrow p2.r \neq \emptyset$; if the Representation $p1.r$ exists then the Representation $p2.r$ must also exist;
- $(p1, p2) \in R2$ iff $e \in p1.r \Rightarrow p2.r \neq \emptyset$; if an element e is present in $p1.r$ then $p2.r$ is not empty;
- $(p1, p2) \in R3$ iff $e \in p1.r \Rightarrow e' \in p2.r$; if an element e is present in $p1.r$ then an element e' must be present in $p2.r$;

- $(p1, p2) \in R4$ iff $e \in p1.r \Rightarrow e \in p2.r$; if an element e is present in $p1.r$ then this element e must also be present in $p2.r$.
- $(p1, p2) \in R5$ iff $e \in p1.r \Rightarrow e' \notin p2.r$; if an element e is present in $p1.r$ then an element e' must not be present in $p2.r$;
- $(p1, p2) \in R6$ iff $e \in p1.r \Rightarrow e \notin p2.r$; if an element e is present in $p1.r$ then this element e must not be present in $p2.r$.

These general rules allow detecting inconsistencies. These rules represent templates which can be instantiated according to the formalism of Lyee like in the following examples.

Examples.

- R2: if there is a Process Route Diagram prd which is not atomic, then there must exist a second Process Route Diagram prd' which must describe it;
- R4: if a Word w is linked to a W02 Pallet, there exists a Scenario Function which produces w ;
- R6: if a Word is produced by a Pallet, then this Word cannot be produced by another.

5. Implementation of the Viewpoint Paradigm to Lyee

We share with researchers in the Requirement Engineering field [2], [5], [7], [9], [10], [11], [15], [22] the opinion that diversity is an unavoidable feature of new software system to be designed. But two statements distinguish the quoted works from our proposition.

At first, Requirement Engineering is generally considered as the earliest stage of a design process. In Lyee methodology and the Viewpoint Paradigm, we try to act still earlier, i. e. on requirements expressed by natural language. The Viewpoint Paradigm dissociates Expression and Content, and thus allows us to consider Expressions which are not provided with formal semantics, like those of natural language.

The second statement is concerned with the relationships between Viewpoints. In many works, the objective is to ensure above all the formal consistency between Viewpoints. We consider that consistency is one possible $\langle E \rangle$ -Correlation, but is far from being the only pertinent one. In fact, the Infometrical Correlation is a tool to manage consistency, as one of the parameters of the convergence state of a requirement process.

Lyee methodology mainly consists of capturing the intention of the user and considers the Requirement Process of the *Object to be* as the expression of intentions; in a complementary way the Viewpoint Paradigm considers the requirement process of the Object as the process producing the sense of this Object. For us it is important to render this process intelligible.

The first step of the VOLYNE project is to validate the presented models in the Lyee and LyeeAll framework. There appears a necessary new skill in the Lyee methodology: the Lyee Requirement Engineer, whose task is to organize the elicitation of the users' intentions. According to the Viewpoint Paradigm, this skill corresponds to a pivotal Actor who will help to provide the Representations corresponding to what must feed the present LyeeAll tool. It is indeed consistent with Lyee methodology and the Viewpoint Paradigm to render all the Actors responsible of their "intentions" towards software to be.

The Requirement Process is iterative until the convergence is observed by the requirement engineer. At the end of the process, the entries of LyeeAll will be complete and consistent in the sense of the Viewpoint Paradigm, and the generation step can be

executed. The question of the computerized support of the models will be studied, in order to integrate it to LyeeAll as an easy to use tool. We intend to support a Viewpoint-Oriented CASE tool for a Lyee Requirement Engineer to the gathering of a complete and consistent set of entries for LyeeAll.

To achieve his objective, three stages must be followed. These stages constitute the three iterative actions the Requirement Engineer must achieve to obtain the expected set of requirements, as shown in Fig. 6.

5.1 Implementation of the Infometrical Correlation: words capture and translation with text classification tools

The first stage is a text mining stage: it implements the Infometrical Correlation to get a written transcription obtained from all the interviews and other forms of needs expression collected during the initial stage of the *words capture* process. The related statistical text processing tools will issue lexical networks in place of classical raw material. The Lyee Requirement Engineer must complete the typology of all the terms issued to characterize the inputs of Lyee and LyeeAll, i.e. *Defined* with their type (screen, database...), *Items* with their attributes (input, output, domain...), *Conditions* and *Formulas*.

5.2 Implementation of the Consistency Management <R>-Correlation: organization of the requirements towards LyeeAll

The entries of LyeeAll are still wrapped in their Viewpoint container. The aim of the consistency management tool dedicated to the Lyee Requirement Engineer is to check the consistency both among the Viewpoints and between the Viewpoints and the Lyee representations of intentions. The inputs of this tool are all the outputs of the previous stage: the tool applies instances of the six R_i rules presented in section 4.2 to deduce the affectation of the *Defined* and *Items* to the *Scenario Functions* and their respective *Pallets*. The rules and the tool which supports them will be helpful for producing all the entries of the automatic program generation of Lyee.

5.3 Implementation of Convergence Indicators Correlation: Convergence Control of the process

The requirement process has to converge towards a state where all the Actors implied in the process agree that the set of collected data is complete and consistent. The third stage of the iterative process help to answer to question like :What are qualitative and quantitative data related to the convergence of the process? How to collect these data? How to render them visible?

We do not constrain the Representations of the Viewpoints to be homogeneous. Actually, either Viewpoints have an autonomous existence before their cooperation, or domain constraints can impose to an Actor of the project to express the Representations of its Viewpoints in a specific formalism. So the Context pole of Viewpoints is defined on two dimensions: Domain, and Time.

The convergence of the process can be viewed by means of a particular <A>-Correlator which defines the state of the cooperation between the different Actors who exert their Viewpoints. We define seven states:

1. isolation: initial state at the beginning of the process;
2. interoperability: the Viewpoints are exerted on the same Object in spite of heterogeneous formalisms;
3. compatibility: the Viewpoints share the Domain dimension of their Contexts;
4. correlation: the Viewpoints are interoperable and compatible;
5. connection: an Actor requests the Representation of several Viewpoints in order to look for inconsistencies, or fusion them; a connection generates a new Viewpoint and <R>-correlator edges in the Graph of Viewpoints between this new Viewpoint and each of the first ones;
6. connectability: two Viewpoints are connectable if there exists at least one path between them in the Viewpoints Graph;
7. convergence: it is a measure of a *distance* between two Viewpoints, exerted by two Actors, this who orders the project and that who manages the project; this distance can rely upon quantitative criteria, such as history of connections between Viewpoints, their number and frequencies.

We define a model for this <A>-Correlator based on a federation of knowledge-based systems. Before LyeeAll code generation, the content of the LyeeAll metamodel database has to be merged with this knowledge based system. The different states of convergence will be notified by logical rules during the requirement process.

Fig. 6. gives a global view of the 3 iterative steps of Lyee Requirement Process according to the Viewpoint Paradigm.

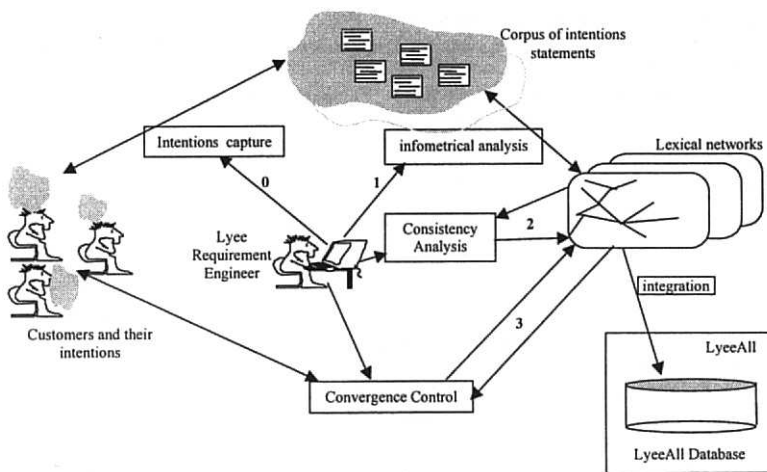


Fig. 6. A global view of the 3 iterative steps of Lyee Requirement Process in the Viewpoint Paradigm

Conclusion

Lyee methodology regards the requirement process for new software from the principle that each actor who is involved in the project expresses his own intentions on the software to be designed. To improve the requirement elicitation process, these actors have been considered from the so-called Viewpoint Paradigm, which deals with the conditions for an object to be designed to acquire sense from multiple sources. This paradigm, whose funding statement is that the sense of an object is the integration of the viewpoints which

are exerted on it, leads us to define the two concepts of Viewpoint and Viewpoint Correlation. Two kinds of Correlations are presented. The first aims at recognising Viewpoints and their Correlations starting from the corpus of intentions statements produced by the users during the requirement process. The second aims at managing the inconsistencies which occur during the process. We claim that the early step of the Lyee methodology can be greatly supported by the Viewpoint Paradigm by providing Lyee Requirement Engineers with appropriate tools that produce the entries required by the code generator of LyeeAll tool.

References

- [1] G. Deledalle, Lire Peirce aujourd'hui, D. Giovannangeli, (Ed), Le point philosophique, De Boeck Bruxelles, 1990.
- [2] S.M. Easterbrook et al., Co-ordinating Distributed Viewpoints: the anatomy of consistency check, *Proceedings of Concurrent Engineering Research and Applications*, West Bloofield, USA, 1994.
- [3] S.M. Easterbrook and B.Nuseibeh, Managing Inconsistencies in an Evolving Specification, *Requirement Engineering '95*, York, IEEE C.S. Press, 1995.
- [4] U. Eco, Segno, Arnoldo Mondadori Editore, 1980.
- [5] A. Finkelstein and H.Fuks, Multi-party Specification, *Proceedings of the 5th Workshop on Software Specification and Design*, Pittsburgh, IEEE C.S. Press, 1989.
- [6] A. Finkelstein et al., Inconsistency Handling in Multi-Perspective Specifications, *IEEE Transactions on Software Engineering* 20, 1994.
- [7] D. Galarreta et al., Study of Dynamic Viewpoints in Satellite Design, *Proceedings of the 9th Symposium IFAC Information COntrol in Manufacturing systems INCOM'98*, Nancy, June 24-26, 1998.
- [8] A.J. Greimas, Sémantique structurale, Larousse, Paris, 1966.
- [9] G. Kotonya and I. Sommerville, Viewpoints for Requirements Definition, *IEEE Software Engineering Journal* 7, 1992, pp. 375-387.
- [10] J.C.S.P. Leite, and P.A. Freeman, Requirements Validation Through Viewpoint Resolution, *IEEE Transactions on Software Engineering* 17 (12), 1991.
- [11] J.C.S.P. Leite, Viewpoint resolution in requirements elicitation, PhD Thesis, University of California Irvine, 1988.
- [12] Introduction to Lyee, 2001.
- [13] E. Morin, La complexité humaine, Champs l'Essentiel, Flammarion, 1991.
- [14] G.P. Mullery, CORE – a method for controlled requirement specifications, *Proceedings of the 4th International Conference on Software Engineering*, München, Germany, IEEE C.S. Press. 1979. pp. 126-135.
- [15] B. Nuseibeh et al., Expressing the Relationships between Multiple Views in Requirements Specification, *IEEE Transactions on Software Engineering*, 20 (10), 1994. pp. 760-773.
- [16] C.S. Peirce, Collected papers, Harvard U.P., 1932.
- [17] L. Perrussel, Expressing Inter-Perspective relationships: A logical Approach, *Proceedings of APSEC '95*, Brisbane, Australia, IEEE CS Press, December 1995.
- [18] K. Pohl, The Three Dimensions of Requirement Engineering, *Proceedings of Software Engineering - ESEC '93, 4th European Engineering Conference*, Garmish-Partenkirchen, Germany, September, 1993.
- [19] J. Rumbaugh et al., The Unified Modelling Language Reference Manual, Addison-Wesley, 1998.
- [20] I. Sommerville and P. Sawyer, Viewpoints: Principles, Problems and a Practical Approach to Requirements Engineering, Lancaster University, Computing Department, Technical Report CSEG/15/1997, 1997.
- [21] I. Sommerville et al., Managing Process Inconsistency using Viewpoints, Lancaster University, Computing Department, Technical Report CSEG/9/1997, 1997.
- [22] P. Zave Classification of Research efforts in Requirement Engineering, *Computing Surveys* 29(4), 1997. pp. 315-321.

Towards an Understanding of Requirements for Interactive Creative Systems

Michael Quantrill
*Creativity and Cognition Research Studios,
Department of Computer Science,
Loughborough University, LE11 3TU, UK*

Abstract. This paper reports on work done at the Creativity and Cognition Research Studios in Loughborough, UK to gain an understanding of the processes and methods of Human-Computer Interaction by users engaged in art practice.

The authors approach to these problems is described. This approach is an extension of drawing practice which uses unhindered human movement within a motion tracked space. Central to this process is the absence of a physical connection between human and computer.

Current work is described that motions towards a definition of requirements for creative, interactive systems. This includes the construction of mechanisms to extend the control of a human in the context of a computer system and the development of a language of articulation that assists the realization of an intention.

A series of experiments have been conducted that explore the relationships between movement and external representations of this behaviour.

1 Introduction

This paper focuses on movement and gesture as creative mechanisms within computer systems and on the impact such research has upon the development of the Lyee [1] methodology. A key aspect of this work is the elaboration of the processes and attributes of interaction as part of creative activity. An unusual approach is taken that uses art practice as a primary means of investigation. The method is described and discussed herein and the problems and benefits of such an approach are given.

The facilities within the Creativity and Cognition Research Studios (C&CRS), Loughborough University [2], include an interaction environment that provides an experimental base for the research described herein. The environment takes body position, movement and gesture as sets of input values to computer systems. It has been shown to be possible to drive computer programs from such input as an alternative or addition to the mouse and keyboard. Human interaction with a computer system using drawing and movement forms the basis for the research activity.

Before further discussion of the work done at C&CRS is presented, the Lyee method will be described and commented upon in relation to this work.

2 The Lyee Methodology

2.1 Theory

Lyee speculates on two worlds, a physical world and a non-physical world. There are 2 sets called A+ and A-. A+ represents atoms from the non-physical or denotative world and A- represents objects that we can cognise in the physical or connotative worlds. Forming a

critical state between the two worlds constructs a mechanism that establishes cognition. Intention arises and forms a set of consciousness atoms which, through a mechanism forge cognition atoms which can then, through a more visible process form observable/cognisable objects in the physical world. Other people in the physical world can also cognise these observable objects. It seems also that each set of consciousness atoms leads to only one set of cognition atoms that in turn lead to only one object. However each set can have subsets that behave in a similar manner to established set theory. This could mean that the intention to draw a line has subsets that have the intention to move the pencil, others that have the intention to move the hand etc.

2.2 Software implementation

The mapping of this theory to software is achieved through a naming convention and the production of structures in software that correspond to some structures defined in the theory, in particular the mapping of the spaces described in the theory to a three dimension like model called TDM. This is a set of axes that map elements at a given temporal position in the process of realising the intention to the coordinates on an axis. In other words if one takes one position of an object with a coordinate from each of the axes, this then represents the realising of the intention frozen at a specific state. Each of these states form, over time, a coupling history. A coupling history is a list of the states and connections made during the process of realising the intention. This also means that each object can be traced back through the coupling history to the intention that spawned the process.

A key aspect of the methodology is that each of these intentions is autonomous and can be serviced independently. Some requirements would however need others to be fulfilled before they themselves can be. For instance, $A=B+C$ could not be satisfied until both B and C have been instantiated. Each of these atomic intentions is mapped in software as scenario functions. The mechanism for servicing all these scenario functions is iterative and the mechanism continually traverses the scenario functions until they are all instantiated. Different routes through the functions are mapped by a Process Route Diagram (PRD). A PRD is established by a software engineer who is experienced in the Lyee method.

2.3 Lyee and Creativity

Lyee has proved to work well with business oriented applications. However the extension and elaboration of Lyee as a tool to assist the development of creative tools is a challenge. How can the current structures of Lyee be adapted or enabled in order to absorb non-business requirements? This is the key question regarding current research and Lyee. In order to address such a question a wider understanding of the requirements of systems that foster creative user interaction is needed. These requirements are being investigated within C&CRS. The first stage of this process is to look at a specific aspect of creativity, which is communication, with particular regard to drawing and movement.

3 Communication and Drawing

Communication occurs in many forms. Mostly this is translated by the mechanisms used to carry it. These mechanisms begin at the moment the desire or impulse to express something in a tangible form occurs: the intention. It is also suggested that methods exist that approach a means of communication that cannot easily be explained, but are undeniable by our experience of them. These methods include the processes of drawing and movement. In fact, drawing is a very significant prototypical example of the use of gesture in expression.

One view of the drawing process is that it is a search for meaning. It can be seen as a conscious investigation of a space that results in transformations of ideas and the emergence of insight. When absorbed in the drawing process, something occurs that is compulsive and

intense. One response is to make marks on paper. However, the making of marks is simply one way of attempting to express the process of transformation that is occurring.

The process of body movement in a space has many similarities with drawing. Drawing is a very immediate process, with a mechanism that often does not depend on explicit instructions or commands. Body movement is similar. As children, we quickly integrate movement into our world and it becomes deeply embedded in our being. We do not utter conscious commands to transfer the energy needed to reposition our bodies. Our minds act by stealth. We see, we feel, we desire and before we are aware, we move. The underlying process of movement is an intensely informative expression and can provide unique characteristic information from those involved.

Focusing on movement as input also means that the participant can be unaware of the physical connection to a computer. They move as they wish. They go where they please within the system. However, the computer is able to record and analyse in parallel with the movement and in reaction to it.

4 Initial system development and inquiry

In order to investigate these concepts, an interactive sensor grid has been constructed. The device is connected to a Silicon Graphics ONYX2 computer that uses IRIX, a version of UNIX, as the operating system. It has extensive graphic tools and has advanced graphics hardware.

The sensor grid consists of a matrix of 16 infra red sensors which combine to give an 8*8 array. These are spaced to confine a 12ftx12ft area. Each sensor set combines a transmitter and a receiver. For each sensor there is a connection via a simple serial wire to a control box. The control box multiplexes these and a microprocessor translates the combined input to 1 output. This output is connected to a Silicon Graphics ONYX2 computer, again via a simple serial cable link. The software to control the sensor box is written in assembler and can be reprogrammed by burning a new ROM. Each of the sensors outputs a signal when the beam is crossed and the control box processes this and sends a 1 Byte data signal to the ONYX2. This contains an x y co-ordinate using the upper and lower 4 bits for x and y respectively.

Connected to one side of the grid is a large data projection screen. This is approximately 8ft*6ft. Image data is back projected from the ONYX2 onto the screen as sensors are tripped. A projector is connected to the secondary graphics output of the ONYX2. This projects an image onto the back projection screen. The image fills the screen. Data is sent down a serial link from the sensor control box to the ONYX2.

The space is configured to enable viewing of the back-projection screen by people within the space. There is also a sound system to enable sounds generated by the programs to be heard from within the space. The arrangement of the tracking equipment does not unduly constrain the space and complete freedom of movement by the user is permitted at all times. Initial experimental programs have been written that perform the following functions:

- Drive the display and audio system and handle the tracking events.
- Translate the user's movement in the plane directly to the viewing plane and provide audio feedback.
- Allow the user to manipulate their position in 3D space even though their movement is in 2D space, a flat plane.
- Allow the user to create and manipulate a set of objects that are placed in modeled space and viewed on the screen. They are also able to alter the properties of these objects.
- Provide timing and colour cues.

The author has previously used computers to investigate issues that relate to technology and art practice [3]. To provide a wider understanding of the current use of technology work done using a device called a Softboard will now be described.

4.1 Earlier Work - The Softboard

The Softboard [4] is a whiteboard (4ft by 3ft) connected to a computer. The whiteboard is similar in design to any conventional whiteboard except it has a laser matrix across its area. There are four colour pens as well as small and large erasers. The laser matrix enables pen and position data to be transmitted to the computer. The application program on the computer looks similar to a drawing package. There is a re-sizeable window, which maps to the physical whiteboard. The resolution is high (4000*3000 points approximately). Any actions made at the physical whiteboard are immediately represented in this window. Together the physical whiteboard and the computer representation are known as the Softboard.

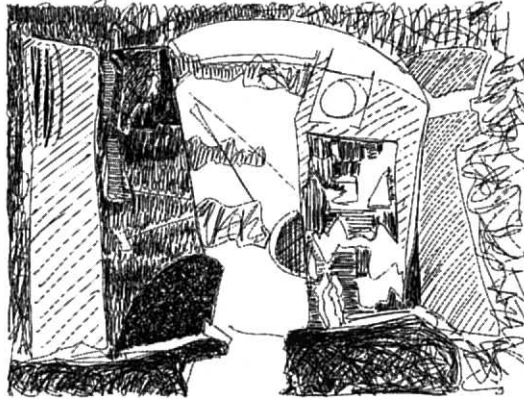


Figure 1: Sequence 3 Page 12 Michael Quantrill 1998 - Output from the Softboard

Drawings are entered onto a "page" using the four pens. A page is one virtual workspace displayed on the monitor. A set of pages forms a sequence. When any mark is made it is recorded as a set of points for the current page. Both positive pen marks and eraser events are recorded. At any time a new page can be generated as a new blank canvas or inclusive of the previous pages marks. The controls for starting and stopping recording and entering new pages are situated both at the whiteboard and within the window on the computer. This enables completion of a whole sequence with pen in hand, never having to touch the computer.

The drawings are physically carried through at the whiteboard. However, due directly to the integration of a computing machine, the creative process is fused with a machine interpretation and so the final piece is an inseparable intertwining of human and machine processes and the works cannot meaningfully be deconstructed, in terms of the artworks they represent, into the human and machine parts. Really, what is going on here is that one process is occurring in a natural, instinctive way, the conscious progression of work from sketchbooks, and another parallel process is occurring "under the surface". This "under the surface" process resulted in a time based dimension, amongst other things, being added to the work that was not expected at the start of the process.

It is important to emphasise that the work with the Softboard does not use an input device obviously designed for a computer, such as a graphics tablet or mouse and that complete freedom of movement is enabled in the space that the work takes place. This means there is little, if any, cognitive load imposed due to the connection to the computer and associated environment. Such freedom of movement enables a creative space to develop that allows the

work to progress without an awareness of the constraints usually associated with electronic media and the need to make allowances for them. This space is a dynamic and integral part of the creative process.

4.2 Feedback

A number of aspects from the Softboard work are significant. One is the fact that a creative space developed. Another is that parallel and augmenting processes occurred. Perhaps, most significant was the growing realisation of the importance of movement.

Drawing is a very immediate process, with a mechanism that often does not depend on explicit instructions or commands. Movement is similar. As children we quickly integrate movement into our world and it becomes deeply embedded in our being. We do not utter conscious commands to transfer the energy needed to reposition our bodies. Our minds act by stealth. We see, we feel, we desire and before we are aware, we move.

The underlying process of movement is an intensely informative expression and can provide unique characteristic information from those involved. Focussing on movement as input also means that the participant can be unaware of the physical connection to a computer. They move as they wish. They go where they please within the system. However, the computer is able to record and analyse in parallel with the movement and in reaction to it. This means that creative functions can be implemented within the computer that are part of the overall system including the participant but without their explicit control. The computer in effect acts by stealth.

One final aspect to consider is trace. Trace can be described as remnant. It may be active but there is some perception of loss, but not with regret. The trace is an indication of previous activity. The loss is due to its receding to the background, similar to memories that fade with time. Interestingly, distant memories can be brought to the present, sometimes with extreme clarity.

Deliberate erasure of the present, the persistent removing of soft marks made on paper for example, is a confirmation that the process is more important than the product. It is also a willingness to allow fragments of the process to play a key role in the output. In this case work is built up as a series of fragments with more or less importance depending on the degree of erasure. Proceeding decisions can then depend both on present values as well as the past.

5 Movement Responsive Interaction

5.1 Considerations

Crucial to the Softboard work was the lack of physical connection to the computer, from the human point of view. There was of course a physical connection via a serial link, but the method of input was using laser-tracked pens so the interface is effectively invisible to the user. Another factor was that freedom of movement within the physical workspace was enabled. Also important was the fact concurrent processes occurred without active and continuous control from the user. These considerations led to the following definition of attributes for the post Softboard work:

- Elements of the physicality of drawing practice should be apparent within the system
- Participants should be free to move in a given area without constraint or hindrance within at least one of the system components

- The computer should be able to collect information not consciously given by the user, which then may become part of the continuing creative process
- Use of the system should inform investigation of the possibilities for integrating human and computing machine activity as part of a creative system
- There should be little cognitive load imposed by the system architecture and function
- Use of the system should be intuitive, notwithstanding basic system orientation
- The user should not be disoriented by the environment
- Some characteristics of the users interaction with the system must be actively used and influence the behaviour of the program
- Use of the system should enable a more clearly focussed discussion regarding the needs of creative users of computer systems, particularly regarding the nature and methods of interaction

At the outset of this investigation it was decided that the system should provide a mechanism to track user movement in real time. It should deliver audio and visual feedback directly related to the sequence of movement that takes place. User contemplation must be permitted at all times by allowing the user (in this case, an artist) to see how their activity is directly affecting some computer process.

5.2 Initial Programs

5.2.1 Program 1

This program maps objects from floor space to screen space in an 8*8 grid. Objects are created as the user moves and their opacity increases as co-ordinates are repeatedly traversed.

5.2.2 Program 2

This program maps 1 object from floor space to screen space in an 8*8 grid. This object moves as the user moves through the space giving the impression it follows the path of the user.

5.2.3 Program 3

In this program the user enters the sensor space. Their position is retrieved from the sensors. An object is displayed on the screen with the co-ordinates of the object mapped to the co-ordinates of the grid. The level of transparency of the object to the background colour is 50%. Objects are red squares on a black background.

When the user moves a new object is created on the screen relative to the users new position. The object at their old position is translated in the z axis. Rather than specify positive or negative z co-ordinates, as they are system dependant, it is sufficient to say the objects appear to move into the screen along the z axis. The distance they move is 1 times their width. The objects cease this type of movement when they have performed this function 9 times for the particular object. From that point on their opacity increases each time a user traverses the objects co-ordinates until the object is fully opaque.

5.2.4 Program 4

This program creates objects based on the user position, but in this program the objects appear to move in and out of the screen along the z axis. This behaviour is set in motion and

controlled in part by the user. The user can always control the object's activity but if the user remains static for a given period of time the objects will move as the user remains static.

5.2.5 Program 5

This program allows the user to create objects and place them both along the x and y axis as well as the z axis. A 3D space is modelled and viewed on the screen. Objects are placed as the user moves in the 2D floor plane. However, through a series of colour and timing cues this floor space is mapped as both an x, y plane as well as an x, z plane. Once objects have been created the user may select (pick) any object and resize it and place it elsewhere on the screen.

5.3 Observations

Program 1 maps the user's movement to screen space and places objects relevant to their floor position. Although fairly simple this provides strong feedback that their movement is directly affecting some process. This is an important factor in associating drawing with the process. Further to this the user is given some control over the properties of the "media". This is enabled by the use of transparency. As the user moves through the space they are able to make subtle, perhaps sensitive changes to the appearance of the objects simply by moving to specific locations within the space. They are able to stop and reflect on what is occurring, which is another key factor in the drawing process.

Each person using the sensor space is not connected by wire to a computer, nor any other physical device. They are able to draw simply by moving. Programs 1, 2 in part, 3 and 5, enable drawing to occur without touching the computer.

In order to track movement and give some feedback a confined space is needed. It was found that the space must be confined to allow the practical viewing of any screen and also to allow reasonable scope for interpreting movement as well as to allow for the limitations of any tracking devices used. This space is sufficiently large so as to allow the user to move freely.

Each program has been designed to permit a different set of feedback information to inform this work. The design is also incremental, with one feature of the space being introduced at a time, culminating in a fairly complex final program.

The polling rate for receiving user events is 10ms. Such a polling rate is necessary to enable little cognitive load due to the system function. Any slower and rapid user movement may cause lag or even data loss as the user moves too quickly for all events to be captured. This would cause unnecessary cognitive load.

The speed of the data gathering and graphics hardware enables the user to easily see that their movement is affecting the display. The use of an 8x8 set of squares allows the user to comfortably track what is happening. There is enough information in the 64 squares to permit the drawing experience and yet not too many to introduce unnecessary complexity. Even in the more complex program 5, although there are more than 64 squares visible, only 64 are available for manipulation and are in focus at any given time. In particular the use of 64 squares that makes the movement and screen co-ordination intuitive. Program 2 takes a different approach, with the square appearing to follow the user. However the single square is still mapped to just 64 possible positions relative to the user's floor position and maintains the intuitive quality of the system.

The visual data, being uniform squares with mostly one colour, is simple. No visual data is presented as decoration. Transparency is a pleasing visual effect but is used here generally in the context of allowing the user to see clearly the density of their movement within the space.

Sound is used sparingly and only at discrete intervals. Where sound is used this is also in response to the movement within the space. Any sound is at a level that is comfortable for the user and clear. As with the visuals the sound should not disorient the user.

Summarising these observations we have:

- The user should receive strong feedback that their activity is affecting some process
- The user should feel in control of the computer system activity in the space
- The user should be able to make subtle and perhaps sensitive changes to the system output, a key concept of much creative art practice
- The user should be able to stop and reflect on the emerging work, another key aspect of creative art practice
- Interaction with the computer system is unencumbered
- A clearly defined, bounded space is needed, but that space should not constrict movement
- The computer system should operate with a minimum response time of 10ms to permit direct correlation of the user activity to system response
- Sound and visual data should not disorient the user

The question now is: How does this work help to define requirements for interactive creative systems using movement and gesture? Whilst requirements are often seen as specific targets that must be met, for example a cost limit, another approach has been developed in which they are expressed as criteria for the evaluation of the system [5]. In an elaboration of this approach, as applied in the specific case of bicycle design, it became clear that these criteria formed the boundaries of the problem space within which the solution to the system design must exist [6]. Thus, the observations listed above can be seen to define the design problem to be solved, or the design space to be searched, if a successful interactive creative system using movement and gesture is to be built. They represent the challenge that any designer, design method or design support system must meet. In many ways, this answer goes back to early discussions of creativity and interaction with computers by Cornock and Edmonds [7], in which no specific requirement or intention appears but, rather, a high level view of the scope of the world of valuable creative systems is articulated.

6 Conclusions

Currently, it is clear the Lyee method works well with business oriented software development. In order to incorporate creativity requirements into Lyee and an understanding of interaction using gesture, these other requirements need to be clearly defined. The programs written so far form part of the investigation, which is defining what such requirements may be.

To implement movement and gesture in a Lyee based system will require the proper definition of requirements for using movement and gesture within a computer system in general. They then need to be matched against the current structures of Lyee. However, it is clear that the current structures are unlikely to be able to be applied in their current form. Requirements for business applications tend to be of a quantitative form and amenable to calculation and unambiguous definition of acceptance or non-acceptance. It seems possible that creative users bring with them harder problems for us to solve. This needs further investigation.

Where creativity is involved and qualitative requirements and attributes are concerned the structures seem less well placed for such requirements to be accepted and met. Therefore, the structures used by gesture-based extensions to Lyee will need to be considered along with the

properties and attributes for interactive creative systems. A framework has been developed within which an investigation into the concepts described in this paper can continue to be elaborated. The key contribution is the development of criteria that it is proposed must be satisfied in the successful development of systems of this kind.

References

- [1] Negoro F. "Principles of Lyee Software", Proceedings of 2000 International Conference of Information Society in the 21 century, pp.441-446, IS2000, Japan, 2000.
- [2] <http://www.creativityandcognition.com>
- [3] Edmonds, E.A. and Quantrill, M.P., "An Approach to Creativity as Process", Reframing Consciousness, Ascott, R. (ed), Intellect Books, Second International CAiiA Research Conference, UWCN, Wales, August 1998, pp 257-261, ISBN 1-84150-013-5.
- [4] <http://www.websterboards.com/>
- [5] Edmonds, E. A. and Candy, L. "Creativity in knowledge work: process model and requirements for support". Proc. OZCHI '95 CHISIG Annual Conference, Ergonomics Society of Australia, 1995. pp 242-248.
- [6] Candy, L. and Edmonds, E. A., "Creative Design of the Lotus Bicycle: Implications for Knowledge Support Systems Research", Design Studies, 17(1), 1996, pp 71-90
- [7] Cornock, S. and Edmonds, E. A. 'The creative process where the artist is amplified or superseded by the computer'. Leonardo, 16, 1973. pp 11-16.

Generating Lyee Programs from User Requirements with a Meta-model based Methodology

Carine SOUVEYET, Camille SALINESI

Centre de Recherche en Informatique, Université Paris 1 - Sorbonne

90, rue de Tolbiac, 75013 Paris - France

{souveyet, camille}@univ-paris1.fr

Abstract: Lyee is a methodology for software development based on the declaration of features of different types. The features are defined according to a typology of pre-defined structures with a well-defined operational semantics directly interpretable by the Lyee program enactment platform. One difficulty with Lyee programming is the complexity of the Lyee-oriented programming constructs. This paper proposes to facilitate Lyee programming by separating the point of view of the user requirements, from the ones of Lyee program structures, and internal coding and enactment of Lyee programs. Each point of view is specified in a separate meta-model. The three meta-models are presented, and their inter-relationships commented and illustrated with the example of a hotel room booking application.

1. Introduction

Lyee is a methodology for software development [1, 2]. Once defined, Lyee programs are executed on a specific platform, i.e. according to a particular systemic. The paradigm underlying Lyee programming is declarative. Programming in Lyee consists in: (i.) declaring the elements composing the program, (ii.) specifying the behavior by declaring values for each element formerly declared, and (iii.) specifying additional pieces of code for those parts not directly taken in charge by the Lyee software execution platform.

The elements declared in Lyee software programs consist mainly in the components of the user interface, and components for interacting with external software (such as databases, file systems, message passing, etc.), and navigation between those.

One major issue with Lyee programming is the difficulty to embrace the variety of types of features and properties to declare. The programmer has to face a number of design/implementation decisions easily driven by the Lyee technology; user requirements are then put together with the technological constraints of the Lyee programming environment. The goal of this paper is to propose methodological elements to guide program construction with Lyee based on user requirements [3]. The strategy taken is based on a clear separation of concerns: on the one hand, user requirements, and on the other hand technological requirements. Each is defined with a meta-model. On the user side, a user requirements meta-model helps defining user requirements independently from any consideration of the Lyee programming environment. On the technological side, the Lyee software development methodology is the main driver of the meta-model. The underlying way of working aims at driving Lyee programming by the user perspective. Thus, the idea is to define user requirements first, then to transform step by step these user requirements by considering aspects of the Lyee technology: the general structure of the program, the user interface, then the interface with external physical components. Each transformation is based on a collection of heuristic rules articulating the meta-models through a third meta-model, called Lyee requirements meta-model, which integrates user requirements, and Lyee program constructs.

The purpose of this paper is to present the three aforementioned meta-models. Section 1 introduces the goals and strategy of the method based on these meta-models. Section 2 presents the three meta-models. Section 3 illustrates these meta-models by applying them on an example (hotel room booking). The user requirements, Lyee requirements and Lyee program are discussed fragment by fragment.

2. Overview of the Meta-models

We propose a methodology based on six meta-models organized in two perspectives and three layers. As Fig. 1 shows, the two perspectives are the one of the *process* and the one of the *product*. In the product perspective, the meta-models are the *User Requirements* meta-model, the *Lyee Requirements* meta-model, and the *Lyee Program* meta-model. Each meta-model belongs to a different layer and defines specific types of concepts. In the process perspective, user patterns, mapping cases and generation rules are defined to create models from user requirements down to Lyee programs: user patterns guide the production of user requirements, mapping cases help transforming user requirements into Lyee requirements, and generation rules automate the generation of actual Lyee code from Lyee requirements.

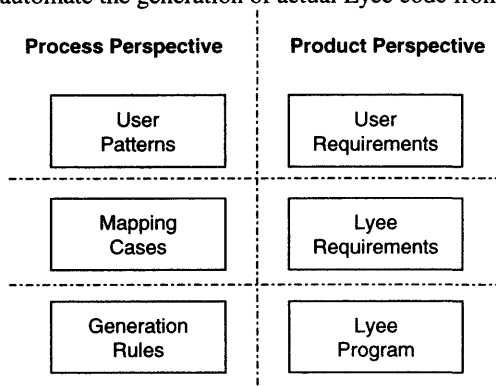


Fig. 1. Organization of the meta-models

The *User Requirements* meta-model defines the concepts for specifying user requirements, i.e. what the user expects from programs: what he/she expects to see, such as the widgets users wants to use, and how he/she expects the programs to behave, e.g. navigation among widgets, calculations, database access, etc.

The *Lyee Requirements* meta-model defines Lyee concepts to specify user requirements from a physical point of view. Instantiating the Lyee requirements model allows for instance to define in Lyee terms the widgets required by users, the navigation among widgets, etc.

The *Lyee Program* meta-model defines the executable concepts of a Lyee program, as they are interpreted by the Lyee program execution platform. Each concept in this meta-model has a well-defined semantics defining the structure and behavior of actual Lyee programs.

The remainder of this paper deals with meta-models in the product perspective. Each meta-model is presented at its turn using the UML notation with a MOF semantics [4] in each of the following subsections.

2.1 User Requirements Meta-model

According to the User Requirements meta-model shown in Fig. 2, user requirements are specified as a graph which *Nodes* identify the features required to support system interactions. Such interactions can occur between the application and users or with external devices, e.g. through widgets and database queries. The graph, called *Precedence / Succedence Graph* (PSG) is structured by *Links* to define how the user wants to navigate between Nodes [3]. According to the meta-model, intermediate Nodes of PSGs are composed of *Items*. This allows specifying the widget components and database variables.

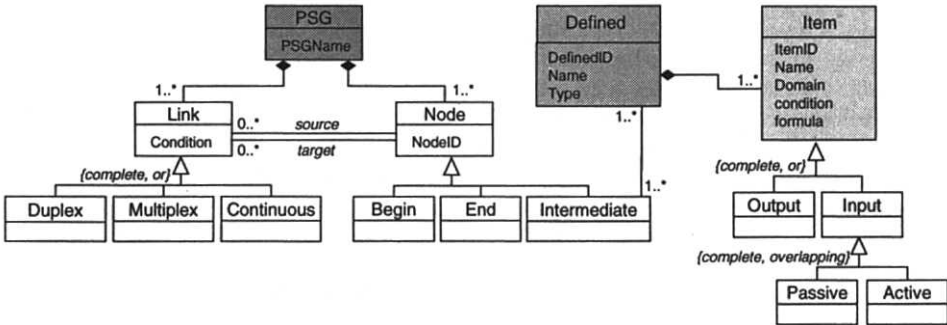


Fig. 2. The User Requirements meta-model

A *PSG* expresses the required overall structure of a single user application. The application name is identified by the *PSGName* attribute. *PSGs* have *Nodes* and *Links*. To each *PSG* Node can be associated data that should be managed by the application, and exchanged between the application and its environment (user, operating system, database, etc). *Links* allow specifying the navigation between *Nodes* when certain conditions are met. *PSGs* are oriented graphs that should have a specific structure: (i) each *Node* and each *Link* should belong to a single *PSG*, (ii) *PSGs* should have a unique *Begin* Node, and a unique *End* Node which respectively identify where the application starts and where it stops, and (iii) any *PSG* should contain a path from its start Node to any of its intermediate Node, and a path from any of its intermediate Node to its stop Node.

Any *Link* belonging to a direct path between an intermediate Node and the start or the End Node is classified *Continuous*. If one excepts the Links to the End Node, the projection of a *PSG* onto Continuous Links should be a tree. Data accessed by the application at the upper level of the tree can be used at the lower levels. Two other kinds of Links are proposed: *Duplex* and *Multiplex*. These Links, when combined with Continuous Links, form loops in the *PSG*. Therefore, they allow to navigate back from one Node to a former Node in the *PSG*. If data from a lower level Node is needed at an upper level then a *Duplex* Link should be used. If the only requirement is to navigate back at a former place of the application (e.g. to go back to a screen), then a *Multiplex* Link should be used.

Intermediate Nodes are containers for *Defineds*. *Defineds* are specified when interactions are required between the application and its external (files and databases are not managed by the application itself, they are thus considered as external to the application). The kind of physical interaction device required by the user is specified by the *Defined* type: screen, file, database, etc. For example, if the user needs to interact with the application to enter data and get information, the *Defined* is a screen. However, if the displayed information is retrieved from a database, a *Defined* of type Database should also be specified. *Defineds* with different types should be specified in separate intermediate Nodes of the *PSG*.

Interactions can be complex: several separate data can be Input by the user through a single widget, the user can require to have data Input and Output through the same widget, a database transaction can correspond to several queries. There is therefore a need to specify the elementary features of interaction between the application and its environment. This is done with *Items* belonging to *Defineds*. A typology of *Items* is proposed: *Input*, *Output*, *Active*, and *Passive*. The value of *Output Items* is expected to come from the application whereas *Input Items* are specified to feed in the application. The behavior expected from the application is to remain still until active *Items* are used. Any *Item* has a domain (numeric, character, button), a *Formula* to indicate how it should be computed if it has to be *Output*, and a *Condition* to specify when it should be computed and *Output*.

2.2 Lyee Requirements Meta-model

User requirements are specified in Lyee terms by *Scenario Functions*. As Fig. 3 shows, *Scenario Functions* are the basic building blocks of Lyee programs. They have a specific structure (a *Scenario Function* is always an aggregate of three 'Pallets') and are composed of *Domain Words* having different behaviors depending on the *Pallet* they belong to. Therefore, each *Domain Word* specifies the elementary behavior of a user requirements *Item*. Overall, the collection of *Scenario Functions* specifying in Lyee terms the user requirements grouped in a *PSG* is called *Process Route Diagram (PRD)*.

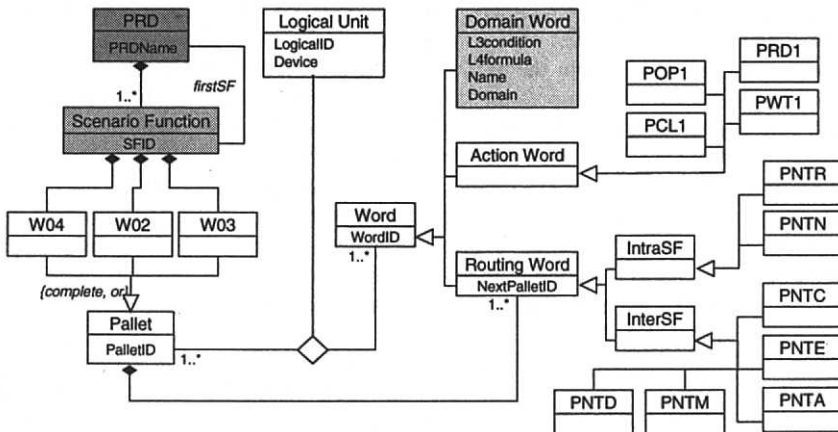


Fig. 3. The Lyee Requirements meta-model

A *PRD* gathers a number of *Words* grouped into *Scenario Functions* and *Logical units*. Each *Word* specifies an elementary feature of the Lyee application. Its behavior depends on its class (*Domain Word*, *routing Word* or *Action Word*), and to its belonging to a specific type of *Pallet* (*W02*, *W03*, or *W04*). The grouping of *Words* in *Scenario Functions* is similar to the grouping of *Items* in *Defined* at the user requirements level. However, this grouping is not direct: (i.) a word can be attached to several *Pallets* of a *Scenario Function*, (ii.) the other way round each *Pallet* of a *Scenario Function* can gather several words, (iii.) a consistent set of words in *Pallets* is called *Logical Unit*, and (iv) each *Logical Unit* corresponds to an *Input* or to an *Output Action* performed by the application through a specific device.

Any *Scenario Function* has a predefined structure based on three *Pallets*, one of each kind: *W02*, *W03*, and *W04*. Each kind of *Pallet* specifies another aspect of the behavior of *Scenario Functions*. *W02 Pallets* specify how *Inputs* occur, *W03* specify internal calculations and navigation between *Scenario Functions*, and *W04 Pallets* specify how

Outputs occur. The basic behavior of a Scenario Function is such that the W04 Pallet is executed first, then the W02 Pallet, then the W03 Pallet.

To each *Pallet* is attached a number of Words. These are of three kinds: Domain Words, Action Words, and Routing Words.

Domain Words are the variables of the application. They can be internal variables specified for calculation purpose, as well as interaction variables specified to collect data from the application environment and prepare data to Output. Domain words can thus be assimilated to the Items specified at the user requirement level. Like these, they have a formula (called *L4formula*) specifying how they should be computed, and a condition (called *L3condition*) specifying when they should be calculated.

Action Words specify Actions to undertake to prepare Input and Output Actions performed by the application. *POP1* and *PCL1* are specified to open and close files. *PRD1* and *PWT1* are specified to prepare the reading and writing of word values on a file, a database, or through screens. *PCR1* and *PCR2* are specified to clear the physical reading and writing areas where word values are Input and Output.

Routing Words specify the navigation between Pallets. When a condition is met, the application shifts to the execution of the Pallet identified by the *NextPalletID*. Two kinds of Routing Words can be distinguished: *IntraSF* and *InterSF*. *IntraSF* Routing Words define the navigation between Pallets belonging to the same Scenario Function, whereas *InterSF* Routing Words define the navigation between Pallets belonging to different Scenario Functions.

IntraSF Routing Words are either *PNTN* or *PNTR*. *PNTN* specify the transition from the W04 Pallet to the W02 Pallet of the same Scenario Function (i.e. to ensure Input after Output), and the transition from the W02 Pallet to the W03 Pallet of the same Scenario Function (i.e. to ensure the computation of internal calculation and evaluation of the next Scenario function to execute). The order of execution of the three Pallets of a Scenario Function is thus W04 (Output), then W02 (Input), then W03 (calculation and routing).

Once the three Pallets of a Scenario Function have been executed, it is necessary to evaluate which is the next Scenario Function. This is ensured by specifying Routing Words in the W03 Pallet. *PNTR* Routing Words specify conditions for proceeding with the execution of the Scenario Function (e.g. because an Input was achieved and an Output has now to be executed). *InterSF* Routing Words specify conditions for proceeding with another Pallet. *PNTC* and *PNTA* specify when to start the execution a following Scenario Function in the PRD. If a *PNTA* is specified, then a return to the current Scenario Function is expected. *PNTE* specify the condition at which the application should stop its execution. *PNTM* and *PNTD* specify the conditions at which the application should go back to a Scenario Function that was already executed, respectively with or without transfer of the data managed in the current Scenario Function.

2.3 Lyee Program Meta-model

According to the Lyee Program meta-model (Fig. 4), a Lyee program contains Pallet Functions aggregating *Signification Vectors* for the execution of Domain Words, *Action Vectors* for the physical reading and writing of data from and to external devices, and *Routing Vectors* for the execution of navigation [5]. Pallet functions belong to *Tense Control Functions*, which implement the Scenario Functions of the Lyee requirements level.

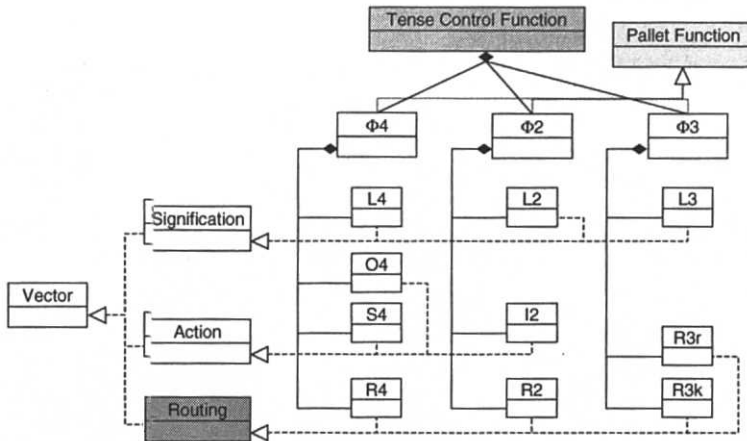


Fig. 4. The Lyee Program meta-model

The behavioral semantics of an actual Lyee program is specified in a typology of Vectors classified into Signification, Action and Routing. Signification Vectors deal with the computation of Domain Words. Action Vectors ensure the preparation of Input/Output Actions. Routing Vectors determine which Signification or Action Vector to execute next.

Vectors are grouped into three kinds of Pallet Functions, Φ_4 , Φ_2 , and Φ_3 , roughly corresponding to the execution of Outputs, Inputs, and internal computations. All the Vectors of a single Pallet Function are executed at a time. $\langle \Phi_4, \Phi_2, \Phi_3 \rangle$ triplets are called Tense Control Functions and noted Φ . These define the operational semantics of Scenario Function as follows:

- The execution of a Tense Control Function starts with Φ_4 . In Φ_4 , the execution order of the Vectors is L_4 , then O_4 , S_4 , and R_4 . L_4 deals with the determination of Output words values. The Action Vectors O_4 and S_4 ensure the preparation and execution of Outputs. The execution of R_4 results in the choice between proceeding with the execution of Φ_4 , and starting to execute the Pallet Function Φ_2 in the same Tense Control Function.
- The execution of a Tense Control Function proceeds with Φ_2 . In Φ_2 , Vector L_2 is executed first to determine for each Input Word whether it should be captured. Then I_2 is executed to capture the value of the Input Words specified in the Scenario Function. This value can come from the user or from an external physical device such as a file or a database. Therefore, the execution proceeds in an asynchronous way. Last, The Routing Vector R_2 is executed to choose between proceedings with Φ_2 (i.e. with Input), or to execute the Pallet Function Φ_3 in the same Tense Control Function.
- The execution of a Tense Control Function ends up with Φ_3 : The execution of Φ_3 starts with L_3 to evaluate the L_3 Conditions of Domain Words. If a condition is met, then an Output should be achieved, and the execution of the current Tense Control Function should proceed. This condition is computed by the R_{3r} Routing Vector ('r' stands for recursive). If the recursion is not achieved, then the R_{3k} Routing Vectors are computed. There are five kinds of R_{3k} ; indeed k stands for 'c', 'm', 'd', 'a', and 'e': R_{3c} and R_{3a} are executed to follow Continuous Links of the PRD (R_{3a} if the Link's target is a Scenario Function followed by a Duplex Link), R_{3m} are executed to follow Multiplex Links, R_{3d} to follow Duplex Links, and R_{3e} are executed to follow a Link leading to the end of the program.

2.4 Summary of the Product Meta-models

Each meta-model sees software from a specific point of view. There are thus correspondences between concepts of the three product meta-models. These are shown in fig. 5.

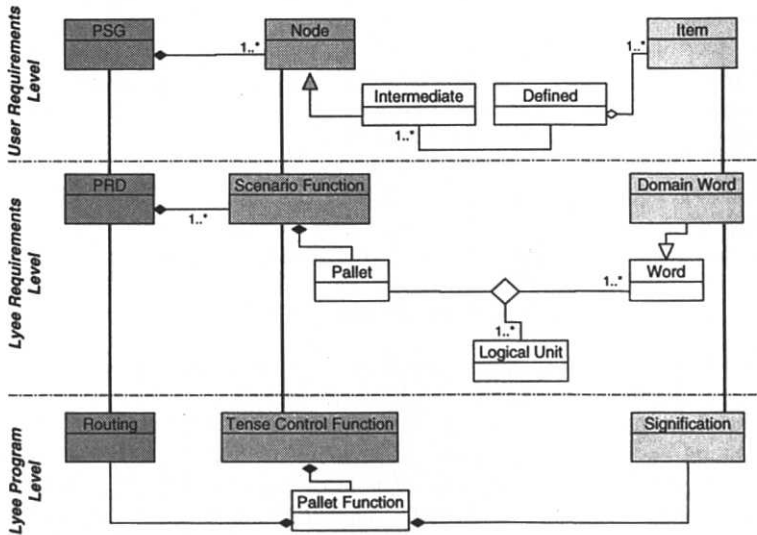


Fig. 5. Overview of the correspondences between the three meta-models

The purpose of the next section is to explore further on these correspondences based on an illustrative example.

3. Example

The chosen case study is the one of a “room booking” system [6]. The system aims at helping hotel customers booking rooms according to their needs. The context of use of this system is the one of a company having agreements with different hotels across a country. The company books rooms at these hotels for its customers and with special prices. The prices depend on the agreements made with the hotel, its category, the required booking dates, etc. Customers are used to contact by phone the agents of the company, express their requirements, be proposed a hotel, and agree on a proposal. Once a hotel is chosen by the customer, a booking is registered, and communicated to the chosen hotel. Customers pay directly to hotels, which pay the booking service back to the company. We shall focus on the room booking registration facility. For the sake of simplicity, the facility only supports the booking of one room at a time.

The system shall let the customer state his/her booking requirements in term of dates (beginning date and ending date of the booking period), as well as hotel location (name of the city in which the hotel shall be), and hotel category (expressed as a number of stars). The system user can be different from the customer; however, the system shall not let unknown customers achieve room bookings. For the sake of simplicity, the system may propose any room available according to the customer’s requirements together with its price for the required period. The system shall not register the booking unless the customer has validated it. A mock-up of the system user interface is presented in Fig. 6. Two screens

are presented: the one for entering the customer's requirements, and the one in which the system makes a proposal which is validated by the customer or not.

The system shall support the following usage scenario: first, the user identifies in the system the customer for whom the booking shall be made, and indicates the booking requirements (booking dates, location and desired hotel category). Then, the system checks in the database if the customer exists and if the required room is available. If the customer doesn't exist or if there is no available room that meets the customer's requirements, a message is displayed. Once an available room has been found by the system, the booking characteristics and its cost are displayed to the user, and recorded into the database as soon as they are validated.

The figure shows two side-by-side system windows. The left window contains input fields for 'Customer ID', 'Beginning date', 'Ending Date', 'Hotel Category (Stars)', and 'City'. Below these fields are 'OK' and 'Quit' buttons. The right window contains input fields for 'Hotel Name', 'Beginning date', 'Ending Date', 'City', 'Room Number', 'Stars', and 'Price'. Below these fields are 'OK' and 'Quit' buttons.

Fig. 6. Two system windows defined in the mock-up to support the required usage scenario

The room booking registration facility is a service provided by the system to build (the room booking system). As required by the User Requirements meta-model, it is therefore specified in a PSG called "PSGBookingRegistration".

The Room Booking Facility can be decomposed into six processing units to support six individual activities:

- the capture of the customer identification and the customer's booking requirements,
- the checking of the customer existence and the availability of a room,
- the display of the proposed booking and its confirmation,
- the display of a "no room available" message,
- the display of an "unknown customer" message, and
- the room booking registration.

Therefore, in addition to its Begin Node (Node1), and End Node (Node8), the PSG has six Intermediate Nodes as Fig. 7 shows. The navigation between these Nodes is specified by Links according to the usage scenario presented in the former section. For the sake of readability, each *Link* is presented by an arrow between its source and target Nodes.

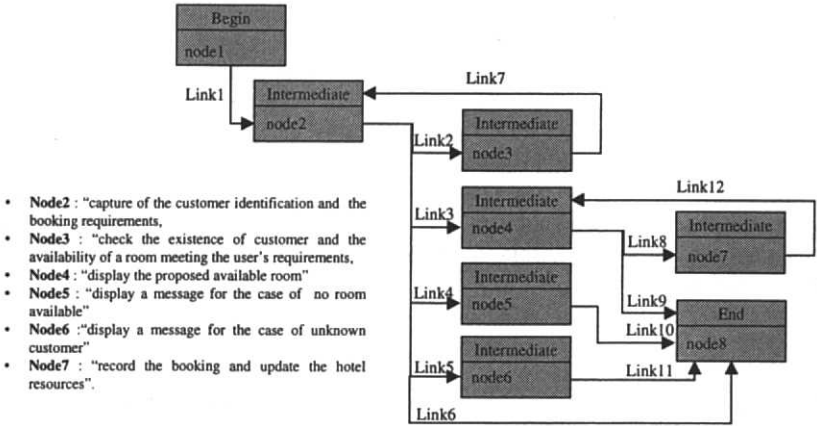


Fig. 7. The “ PSGBookingRegistration” PSG

4. Illustration of the Meta-Model Equivalences through the Three Layers

This section illustrates the three meta-models and the correspondences between them with the “room booking” system example. The following shows how the system is developed by instantiating the three meta-models, and emphasizes the corresponding meta-model instances.

At the Lyee requirements level, the “ PSGBookingRegistration” *PSG* is described with a *PRD* (Process Route Diagram). This *PRD* is modeled as an aggregate of one or several *SF* (Scenario Functions). The *PRD* is implemented in the Lyee program structure by one *Tense Control Function* (Φ) for each *SF*. Besides, *R3k Routing Vectors* ensure at the Lyee program level the branching from one *SF* to another. Fig. 8 shows that at the Lyee Requirements Level, the *PSG Begin Node* is taken into account by the instantiation of the “firstSF” association between the *PRD* and the “*SFCaptureReq*” Scenario Function. At the Lyee program Level, this Scenario Function is executed first by invoking the first *Tense Control Function* in the program. i.e. the one that corresponds to it.

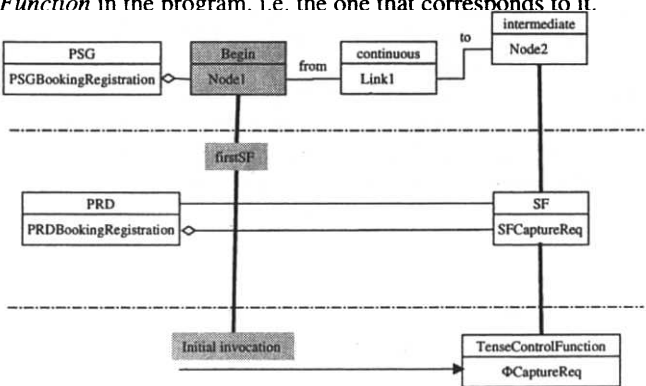


Fig. 8. Correspondences with the *Begin Node*

There are several ways to terminate the program: after the booking validation, after the display of the error messages, and after the capture of the customer requirements, if the user asks for a cancel. Such program terminations are specified in the PSG by Links between the Nodes after which the program terminates (i.e. Node2, Node4, Node5, and Node6) and the

End Node (Node8). At the Lyee Requirements Level, this is specified by PNTE Routing Words. These are put in the Scenario Functions corresponding to the Nodes after which the program terminates (e.g. SFCaptureRFeq for Node2). At the Lyee program Level, the program ends when a R3e Routing Vector affects and “End” flag to the NexPalletID property of the involved Routing Word. Then, the active $\Phi 3$ Pallet Function ends as well as the active Tense Control Function (Φ), and no other Tense Control Function is executed.

The program is expected to achieve a number of processing: Input, Output, and calculations. These are identified in the PSG through the intermediate Nodes. For example, Node4 presented in Fig. 9 corresponds to a calculation (of the price of the room), to an Output (through the display of the booking proposal), and to an Input (through the capture of the customer confirmation). At the Lyee Requirements Level, PSG intermediate Nodes are specified with Scenario Functions such as “SFProposal” for Node4. Each Pallet of the Scenario Functions has a specific purpose; in “SFProposal”, W04 deals with the calculation of the price of the room and with the Output of the booking proposal, and W02 deals with the Input of the customer confirmation. At the Lyee program Level, the $\Phi 2$ and $\Phi 4$ Pallets Functions ensure respectively the actual achievement of Inputs and Outputs. The execution of the calculations is partly managed by $\Phi 3$ and $\Phi 4$.

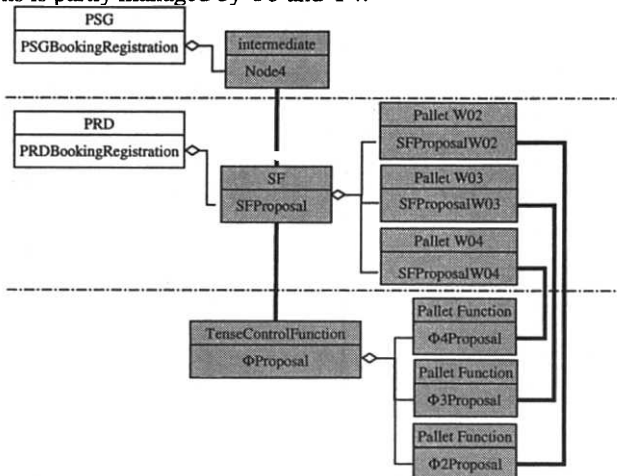


Fig. 9. Correspondences with the *Intermediate Node Node4*

Two aspects of the required interactions between the system and its external environment can be considered:

1. these interactions can occur either between the program and the user through screens (e.g. to capture the customer's booking requirements or to display error messages), or between the program and other physical components such as database or file system (e.g. to check the customer existence, or to register the booking).
2. these interactions are either Inputs (such as data capture by the user in the system, and data retrieval from a database), or Outputs (such as data display to user and data record in database).

The kind of required physical interaction device (screen, file, database) guides the identification of Defineds in PSG intermediate Nodes. If different interaction devices are to be used in combinations, then different Defineds should be specified, and associated to different intermediate Nodes. For example, the customer existence is checked after the customer Id has been captured. These correspond respectively to a database query, and to a user interaction through a screen. Therefore, two Defineds should be specified, each associated to a different Intermediate Node (Node3, and Node2).

As mentioned earlier, Node3 has two purposes: (i.) to check the customer existence, and (ii.) to check the availability of a room. These are achieved by database queries. Each physical access to a database has a purpose; this should be reflected in the PSG by the association of one Defined for each database access. Therefore, as Fig. 10 shows, two Defineds are associated to Node3: one for checking the customer existence (dbcustomer) and one for checking the availability of a room (dbroom). At the Lye Requirements Level, the intermediate Node is specified by a Scenario Function, and the Defineds by Logical Units. There are for example two Logical Units associated to the SFCheck Scenario Function: FR02, and FR03, which respectively correspond to customer checking and room availability checking. These Logical Units are specified with POP1, PRD1, and PCL1 Action Words to deal with the opening, physical reading and closing of the databases. At the Lye program level, a Tense Control Function executes the SFCheck Scenario Function, and an Action Vector executes the FR02 and FR03 Logical Units. These Logical Units correspond to database queries; an I2 Vector thus executes them. If these Logical Units had corresponded to a database update an O4 Vector would have executed them.

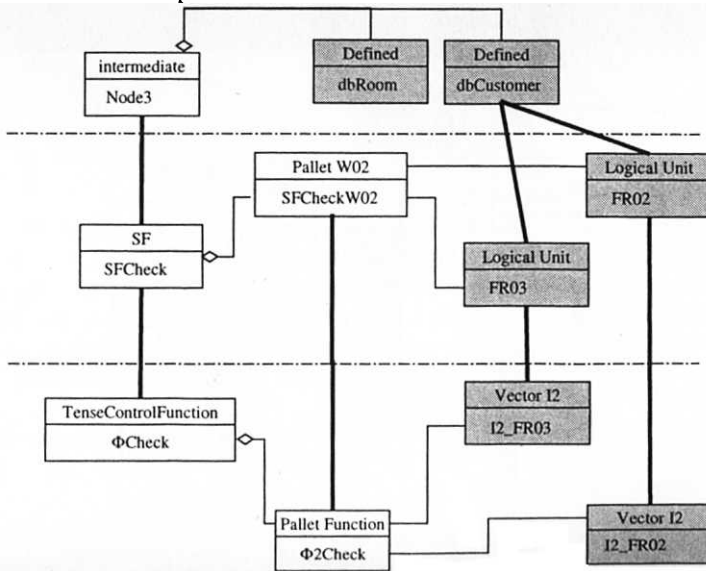


Fig. 10. Correspondences with the Defineds “dbCustomer” and “dbRoom”

Contrary to Node3, Node2 has only one purpose: to capture the customer’s requirements. Therefore, as shown by Fig. 11, only one Defined is associated to this Node (CaptureScreen). However, two Logical Units specify this Defined at the Lye Requirements level, one for the user Input (SR01), and one for the Output to the user (SW01). The former Logical Unit is specified with a PRD1 Action Word, the latter with a PWT1 Action Word. Like for database access, user Inputs are physically executed by I2, and Outputs by O4.

To sum up there are seven Defineds in the PSG:

- CaptureScreen: to get the CustomerID, BeginDate, EndDate, Location, and HotelCategory, and with two buttons for validating and canceling.
- dbCustomer: to check the customer existence based on the CustomerID.
- dbRoom: to check the availability of a room based on the BeginDate, EndDate, Location and HotelCategory, and to get from the database the HotelName, RoomNumber and DailyPrice of the available room.

- **ProposalScreen**: to display **HotelName**, **Booking dates**, **Location**, **Room number**, **Hotel Category**, **Price of the room** and two buttons for validating the booking or canceling the whole transaction.
- **CustomerErrorScreen**: to display a generic error message and return to the **CaptureScreen** Defined by a return button.
- **NoRoomScreen**: to display a message indicating that no room is available and return to the **CaptureScreen** Defined by a return button.
- **dbBooking**: to register the booking with the **CustomerID**, **BeginDate**, **EndDate**, **HotelName**, and **Room number**.

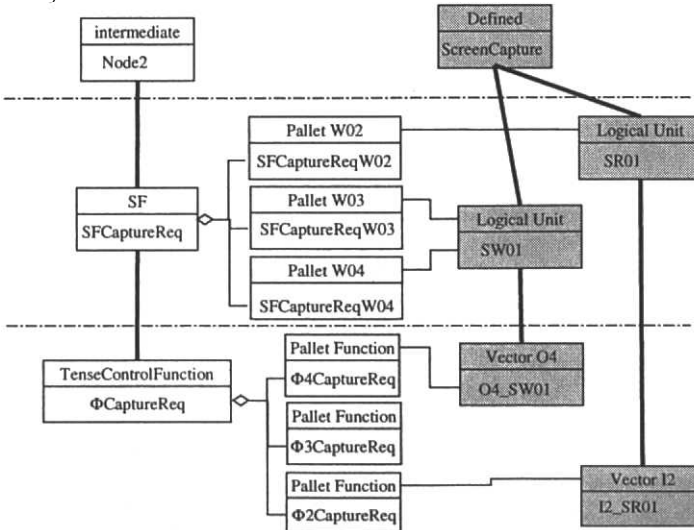


Fig. 11. Correspondences with the “ScreenCapture” *Defined*

The User Requirements meta-model indicates that **Defineds** are Item aggregates. For instance, the **CaptureScreen** **Defined** contains seven Items: **CustomerID**, **BeginDate**, **EndDate**, **Location**, and **HotelCategory**, **CmdOk**, and **CmdQuit**; four Items should be specified in the **dbRoom**, one for each parameter of the query. Some of these are used for **Input** (e.g. all the Items of **CaptureScreen** and **HotelName** in **dbRoom**), and others are used for **Output** (e.g. **RoomNumber** in **ProposalScreen**, and all the Items of **dbBooking**). **Input** Items can be **Active** or **Passive**. For example, the **CmdOk** and **cmdQuit** Items inform the program that the user expects that he proceeds with the execution; these are thus **Active**. On the contrary, the **CustomerID**, **BeginDate**, **EndDate**, **Location**, and **HotelCategory** Items of the **CaptureScreen** **Defined** do not trigger anything in the application; they are thus **Passive**.

As illustrated in Fig. 12, one **Domain Word** is specified at the Lyee requirements level for each Item. Each **Domain Word** belongs to a **Pallet** and a **Logical Unit**. **Domain Words** corresponding to **Input** Items (e.g. **DailyPrice**) are in the **W02** **Pallet**; **Domain Words** that correspond to **Output** Items (e.g. **Price of the room for the booking period**) are in **W03** and **W04** **Pallets**. The affectation of **Domain Words** to **Logical Units** depends on the direction supported by the corresponding Item, and on the device (screen or database) used with the Item. **Domain Words** corresponding to **Active** Items have the **Domain** property set to ‘K’. The **Domain** property of all the other **Domain Words** has the same value as in the corresponding Items.

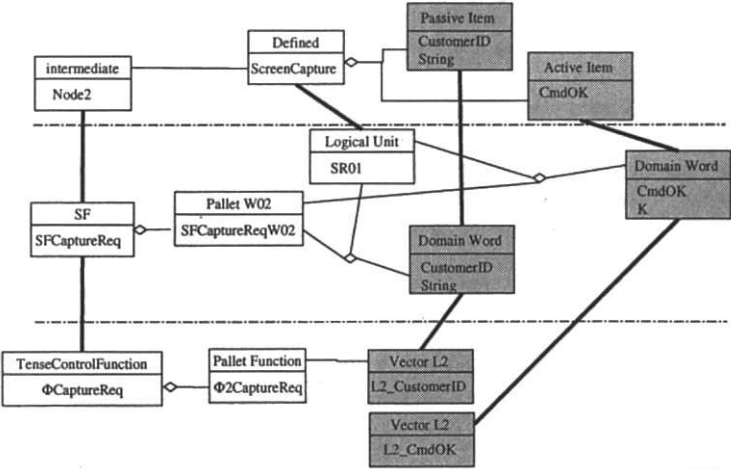


Fig. 12. Correspondences with the *Items* “CustomerID” and “CmdOK”

At the Lvee program Level, the $\Phi 2$ Pallet Function is in charge of executing Input; $\Phi 3$ and $\Phi 4$ Pallet Functions deal with Output.

In $\Phi 2$, one L2 Signification Vector is executed to determine the value of each Domain Word corresponding to an Input Item. For example, the value of the Domain Words specifying each Item in the CaptureScreen Defined is obtained by the execution of an L2 Signification Vector. This holds as well for active Items such as buttons. The value of the corresponding Domain Words is set to ‘true’ when then button is pressed, whereas it is ‘false’ as long as the button is unused.

In $\Phi 3$ and $\Phi 4$, one L3 Signification Vector and one L4 Signification Vector are executed to respectively control if the value of a Domain Word corresponding to an Output Item should be computed, and to compute it. For example, Fig. 13 shows that the “Price of the Room” Item has a formula to compute the price of the room based on a daily price and the number of days in the booking period. This formula should always be computed when the program makes a proposal because the Item’s condition is set to ‘true’. The Item’s condition and formula are respectively specified in the L3Condition and L4formula of the corresponding Domain Word. The former is evaluated by the execution of the L3 Signification Vector, the latter is computed by the execution of the L4 Signification Vector.

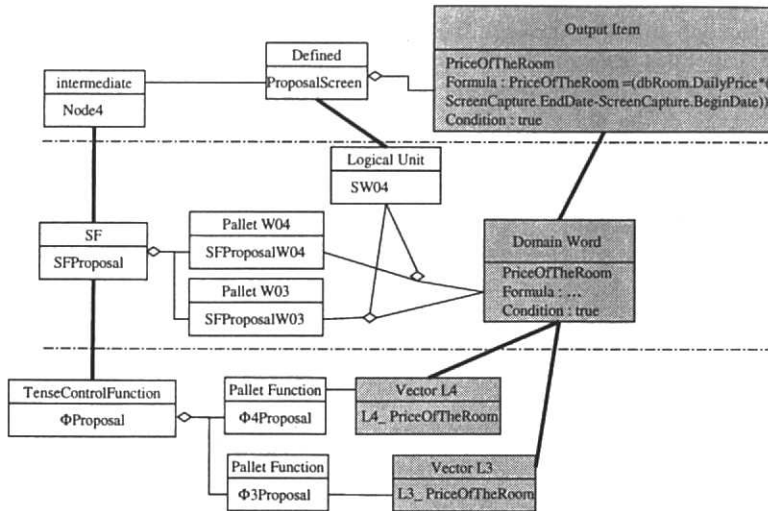


Fig. 13. Correspondences with the “Price of the Room” Item

The execution of the L2 and L4 Signification Vectors is tightly related to the execution of the I2 and O4 Action Vectors. While I2 performs the physical reading from screen, database etc., L2 affects the logical value to the corresponding Domain Word. The other way round, the value of Domain Words is logically computed by L4 before it is physically displayed on a screen or written on a database by O4.

The navigation between the Nodes of the “PSGBookingRegistration” PSG is described by the twelve Links presented in Figure 6. Continuous Links are used to move forward in the application:

- from one screen to another (e.g. Link3 allows to navigate from the customer requirements capture screen to the display of the booking proposal if the customer exists and an available room was found),
- from screen to database (e.g. Link2 to check the existence of the customer after having identified his/her customerID in the customer requirements capture screen), and
- with Begin Node or End Node (e.g. Link9 leads to the end of the program after the booking is registered, Link1 leads to the customer requirements capture screen at the Beginning of the program execution).

The PSG also contains two backward Links: Link7 to return from Node3 to Node2, and Link12 to return from Node7 to Node4. These Links are Duplex Links since they allow to transfer data from the source Node to the target Node. For example the customerID, HotelName, RoomNumber and DailyPrice have to be returned from Node3 to Node 2 if they exist. This requirement is identified by the Link7 Duplex Link.

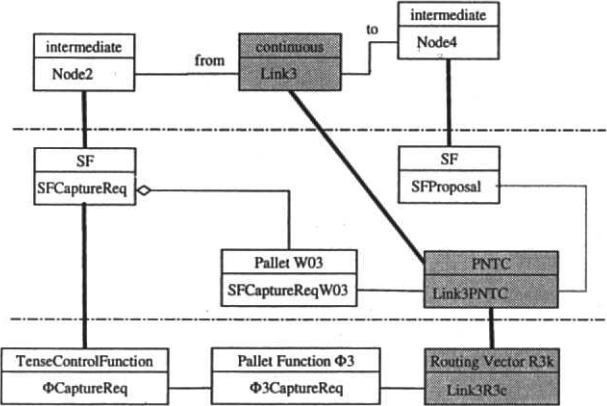
Continuous Links are specified at the Lyee Requirement level by PNTC, PNTA, and PNTE Routing Words. Duplex and Multiplex Links are respectively specified by PNTD and PNTM Routing Words. For example:

- Link3 is specified by a PNTC, because it is a Continuous Link,
- Link2 is specified by a PNTA, because it is a Continuous Link followed by the Link7 Duplex Link,
- Link7 is specified by a PNTD because it is a Duplex Link, and
- Link9 is specified by a PNTE, because its target is the PSG End Node.

These Routing Words are systematically associated to the W03 Pallet in the Scenario Function specifying the Source Node of the Link it corresponds to. Fig. 14 shows for example that the Source Node of Link3 is the “CaptureScreen” Node, the corresponding

Scenario Function is SFCaptureReq. The aforementioned PNTC Routing Word is thus in the W03 Pallet of this Scenario Function.

As indicated in the meta-models, these Links are followed by executing the R3k Routing Vectors of the $\Phi 3$ Pallet Functions that belong to the Tense Control Function executing the Scenario Function where the Routing Words are specified.



5. Conclusion

The work presented in this paper aims at guiding the generation of Lyee programs. The approach is based on meta-modelling, which has been widely used in Information System engineering to aid in the definition and understanding of methods. Three meta-models are proposed: (i.) the User Requirements meta-model, to help identifying the user requirements independently from the Lyee programming structure, (ii.) the Lyee Requirements meta-model, to specify user requirements in the Lyee perspective, and (iii.) the Lyee Program meta-model, to define how the user requirements are achieved by Lyee program execution.

The use of three proposed meta-models is illustrated with the ‘room booking’ case study. The purpose of this case study is not only to indicate how the meta-models instantiate, but also to explore the multiple relationships between the meta-models. This gives a first hint on how to generate a Lyee program starting from user requirements. However, we believe this is not enough: the three proposed meta-models are meant to be used in combination with user patterns, mapping cases and generation rules to respectively guide the instantiation of each of them. The next step in our research program is to develop these guidelines based on the relationships we have experienced between our three meta-models. The result would be a complete method including a formal definition of both the products and the processes for Lyee programming.

References

- [1] F. Negoro. *Methodology to Determine Software in a Deterministic Manner*. Proceedings of ICII, Beijing, China, 2001.
- [2] F. Negoro. *A proposal for Requirements Engineering*. Proceedings of ADBIS, Vilnius, Lithuania, 2001.
- [3] C. Rolland. *Understanding the Lyee Methodology through Meta-Modelling*. Proceedings of 6th World Multi Conference on Systemics, Cybernetics and Informatics. Orlando, Florida, USA, July 2002.
- [4] Object Management Group. *MOF Specification*. OMG Document N° ad/97-08-14, September 1997.
- [5] C. Souveyet, C. Rolland, R. Kla. *Tracing the Lyee Execution Process*. E-Lyee Project, report N° TR1-3. October 2001.
- [6] C. Salinesi, M. Ben Ayed, S. Nurcan. *Development Using Lyee, a Case Study with LYEEALL*. E-Lyee Project, report N° TR1-2. October 2001.

Lyee¹ Program Execution Patterns

Mohamed BEN AYED
Université Paris1 Panthéon Sorbonne
CRI, 90 Rue de Tolbiac
75013 Paris, France
mohamed.benayed@malix.univ-paris1.fr

Abstract. The research undertaken aims to find regularities in Lyee program execution traces and to relate the trace chunks to well defined types of Lyee design situations. Our findings take the form of Lyee execution patterns, each pattern coupling a situation with a trace chunk. The paper presents and illustrates them with an example.

1. Introduction

Lyee-Sorbonne² is a Franco-Japanese research project aiming at developing a methodology that supports software development in two steps, requirements engineering and code generation. The former is the contribution of the Sorbonne's group whereas the latter is provided by LyeeALL.

The overall objective of the research activity of the Sorbonne Unit is to apply a *method engineering* approach to the Lyee methodology [5,6]. The expected results consist on formalising the product and the process aspects of the Lyee method, proposing improvements to guide the process and to contribute to the extension of the functionality of LyeeALL accordingly.

As a prerequisite of this method engineering activity, we developed a number of application examples using the Lyee method and the support of LyeeALL. This paper is a reflection on these experiments. It takes the form of a set of *execution patterns* that we discovered by analysing the program traces of the generated code for these examples. As any pattern, an execution pattern associates a *problem* to its *solution*. The problem is a *design situation* typical of the Lyee way-of-thinking whereas the solution is a *sequence of steps* of the Lyee executed program. We identify *eight patterns* that, we think, are sufficient to explain any Lyee program trace. In other words, any Lyee program execution trace is an aggregate of several of the 8 pattern traces.

The patterns show that there are regularities in the Lyee program execution control. Moreover, they allow us to relate a typical sequence of steps in program execution control to a well identified design situation. This was helpful to understand the key design situations to focus on during the requirements acquisition phase [9].

The paper presents the eight execution patterns and illustrates them with an example. The rest of the paper is organised as follows. Section 2 introduces the notion of Lyee program execution trace and the notations used to present a program trace in the patterns. Section 3 presents the *execution patterns* which are illustrated with an example in section 4.

¹ Lyee, which stands for Governmental Methodology for Software Provi~~dence~~, is a methodology for software development used for the implementation of business software applications. Lyee was invented by Fumio Negoro.

² This paper hereof is contributed to the Lyee International Collaborative Research Project sponsored by Catena Corp. and the Institute of Computer Based Software Methodology and Technology.

2. Tracing a Lyee Program Execution

In this section we first provide a brief overview of the Lyee program structure and execution control and then, introduce the notations to describe a Lyee program execution trace.

2.1. Lyee Program Execution Control

The Lyee approach and its support tool LyeeALL aim at transforming software requirements into code. The essence of the approach is to reduce software requirements to the description of *program variables* called *words*, and to generate the *control structure* that logically processes these variables and produces the expected result. Despite the traditional design approaches in which both the variables and the control structure of the program must be designed, LyeeALL generates the latter provided an appropriate description of the former is given. The underlying Lyee approach comprises an *original framework* to structure programs, an *engine* to control their execution and a *generation mechanism* to generate programs from given requirements.

The Lyee engine controls the determination of words by executing a function, the *Tense Control Function* on a program generated by LyeeALL. This program is an instance of a generic model, the *Process Route Diagram (PRD)*. The PRD provides the structural framework of Lyee programs whereas the *Tense Control Function* ensures the dynamic control of program execution.

As shown in Figure 1, the structure of a Lyee program is a directed graph whose nodes are instances of a basic structure called *Scenario Function*. A *Scenario Function (SF)* is itself an aggregate of three sub-structures, called *Pallets*, *W04*, *W02* and *W03*. Each pallet structure is further decomposed into sub-structures called *Vectors*. There are three types of vectors, *Signification Vectors (Li)*, *Action Vectors (I2, O4, S4)* and *Routing Vectors (Ri)*. Besides, there is a specific set of vectors for each pallet; for example pallet *W04* has four vectors, *L4*, *O4*, *S4* and *R4*. Finally each vector has an algorithmic structure which is a specialisation of the generic structure called the *Predicate Structure* [7].

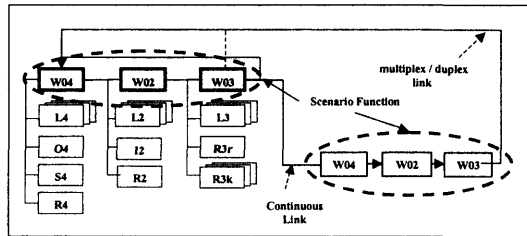


Figure 1 : Lyee Program Execution Control

The *Tense Control Function* ensures the program control through *Pallet Functions*. The control is summed up in the following formula [6]:

$$T_U = \Phi[\Phi4(\{L4,j\}, \{O4, \alpha\}, \{S4, r\beta\}, R4) + \Phi2(\{L2,i\}, \{I2, \alpha\}, R2) + \Phi3(\{L3,j\}, \{R3,k\})].$$

As expressed by the formula, the Tense Control Function Φ may be defined as $\Phi4 + \Phi2 + \Phi3$. In short, $\Phi4$ controls the determination of output words and their physical presentation on a device such as a screen, $\Phi2$ ensures the capture of input words whereas

$\Phi 3$ aims at verifying the conditions under which the determination of output words is possible. Each pallet function controls the program execution thanks to its related vectors. It shall be noticed that the Routing Vectors are used to hand over the control either from one Pallet to another in the same SF or to another SF. The Routing Vectors R4, R2 and R3r allow to progress locally from one Pallet Function to another one in the same SF. Contrarily R3k Routing Vectors are used to hand over the control to another SF. There are three types of SF links which are handled by the Routing Vectors R3k : continuous, duplex and multiplex links. The former is a forward link whereas the two latter are backward links. A duplex link is required when words generated by the dependee SF must be transferred to dependent SF that starts its execution by Pallet Function $\Phi 3$. The multiplex link holds in case of no data transfer and the SF execution starts with Pallet Function $\Phi 4$.

2.2.Lyee Program Execution Trace

The purpose of *program execution tracing* is to provide an exact image of the sequence of steps by which the program transforms a set of inputs in a set of outputs. In the case of Lyee, the program execution trace has a predefined structure which is the one underlying the program execution control, namely the Tense Control Function. In fact, a Lyee program execution trace is an instance of the T_U formula introduced above. Each step of the trace is an instance of the T_U formula element, e.g. L4.w for a step which produces the output word w, R3r to hand over the control to the $\Phi 4$ of the same SF, L2.x for acquiring the input word x, etc...

In the following, we present a Lyee program execution trace as a sequence of instantiated vectors. Figure 2 is an example of trace based on the notations summed up in Table1.

Notations	Meaning
Li. word	Instantiation of the Significant vector Li for "word"
Φa Lb	Execution of the pallet function Φa for vector Lb
I2	Input vector for pallet 2
O4	Output vector for pallet 4
S4	Structural vector
R4	Routing vector for pallet 4
R2	Routing vector for pallet 2
R3r, R3k	Routing vector for pallet 3: - r : recursive link - k: continuous, duplex or multiplex link

Table1 : Notations for Lyee Program Execution Tracing

As illustrated in Figure 2 below, the trace is presented in a table with three columns. The first indicates the step number, the second one provides the trace by reference to the vector executed at this step and the third one explains the effect of this execution. For instance in Figure 2, step 1 corresponds to the evaluation of the output word, *word1* by the signification vector L4; in step 2 the O4 execution leads to display the screen, and step 4 hands over the control to Pallet W02.

Step	Trace	Effect
(1)	$\Phi 4$ L4.word1	None
(2)	O4	Display screen
(3)	S4	Clear memory word
(4)	R4	NextPallet Id = SF01W02
(5)	$\Phi 2$ L2.word2	None
(6)	I2	None
(7)	R2	NextPallet Id = SF01W03
(8)	$\Phi 3$ L3.Goal	None
(9)	R3r	NextPallet Id = SF01W04

Figure 2 : Trace Example

3. Execution Patterns

This section presents the eight Lyee program *execution patterns*. Patterns have been introduced in software engineering as a mean to capture a *solution* applicable to a *problem* in a given *context*. A pattern identifies a problem which occurs again and again and provides a generic solution that can be recurrently applied to solve the problem. In [1], Alexander defines a pattern as describing “a problem which occurs over and over again in our environment and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice”. A large number of similar definitions of the term “pattern” exists today [2,3,4]. All these definitions share two main ideas. First, a pattern relates a recurring *problem* to its *solution*. Second, each problem has characteristics that distinguish it from other problems.

3.1. Lyee Execution Pattern Structure

The Lyee *execution patterns* presented in this section aim at identifying *sequences of steps* which occur again and again in the execution of Lyee programs. Each pattern associates a *sequence of steps* to the *situation* in which this sequence is executed.

A *sequence of steps* is expressed using the trace notations introduced before. Every of the *sequence of steps* of the eight patterns can be found as parts of a SF execution.

The corresponding *situation* shall thus characterise the SF in some way. To achieve this objective, we propose to view any SF as a *state transition* (Figure 3) that transforms a set of words (W_{before}) into another set of words (W_{after}), provided some input words (W_i). A pattern situation characterises the transition through conditions expressed on the three types of words W_{before} , W_i and W_{after} .

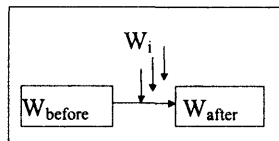


Figure 3 : The semantic view of a Scenario Function

The state *before* is defined by the set of words called W_{before} that, if not empty, has been determined by previous executions of one or several SFs. The state *after* is defined by a set of new input and/or output words called W_{after} resulting from the SF execution. As shown in Figure 3, the transition requires some input words called W_i from the external world (user world or database world). Words of W_i can be captured in one shot or in several asynchronous steps. This refers to the fact that the transition may encompass one or several interactions with the external world. In other words, the SF is related to a state transition

including at least one interaction with the external world but may comprise several ones. We will see later on the influence of this on the program trace.

Every pattern is presented in the following in two parts (a) the *situation* in which it is applicable and, (b) the typical *sequence of steps* of program execution corresponding to this situation. In each part, the situation and the sequence of steps respectively, are presented and then, commented.

3.2. Pattern P1: Start of an Independent Scenario Function

Situation: $W_{\text{before}} = \emptyset$

We qualify this SF as *independent* because its execution does not depend on previous SF executions.

Execution trace :

Step	Trace	Effect
(1)	$\Phi 4.\{L4.j\}$ j : set of output words	None
(2)	$\Phi 4.O4$	Display the screen / None
(3)	$\Phi 4.S4$	Clear word memory / None
(4)	$\Phi 4.R4$ <i>Wait for the user interaction (asynchronous relation between pallets W04 and W02)</i> Or <i>No Wait (synchronous relation between pallets W04 and W02)</i>	NextPallet Id=SF01W02

Table 2 : Execution Trace of Pattern P1

The main characteristic of the trace presented in Table 2 is that one execution of the $\Phi 4$ pallet function is sufficient to trigger a successful execution of the $\Phi 2$ pallet function (in the sense that it allows capturing the input). The $\Phi 4$ pallet execution leads to displaying the empty screen (in the case where the SF is typed screen) and to clear the word memory.

3.3. Pattern P2: Start of a Dependent Scenario Function

Situation: $W_{\text{before}} \neq \emptyset$

We qualify this SF as *dependent* because the SF execution depends of words determined previously by the execution of other SFs.

Execution trace :

Step	Trace	Effect
(1)	$\Phi 4.\{L4.j\}$ j : set of output words	None
(2)	$\Phi 4.O4$	None
(3)	$\Phi 4.S4$	None
(4)	$\Phi 4.R4$	NextPallet Id=SF01W02
(5)	$\Phi 2.\{L2.i\}$ i : set of input words	None
(6)	$\Phi 2.I2$	None
(7)	$\Phi 2.R2$	NextPallet Id=SF01W03

(8)	$\Phi 3.\{L3,j\}$ j : set of output words	Determine the necessary conditions of output words based on words of previous SF
(9)	$\Phi 3.R3r$ r : recursive link	NextPallet Id=SF01W04
(10)	$\Phi 4.\{L4,j\}$ j : set of output words	Determine the output words based on words of previous SF
(11)	$\Phi 4.O4$	Display the screen / None
(12)	$\Phi 4.S4$	Clear memory word / None
(13)	$\Phi 4.R4$	NextPallet Id=SF01W02 Wait for the user interaction (asynchronous relation between pallets W04 and W02) Or No Wait (synchronous relation between pallets W04 and W02)

Table 3 : Execution Trace of Pattern P2

The trace presented in Table 3 shows that the words of the state *before* can be determined only at the second execution of the pallet function $\Phi 4$. The first iteration of the sequence of pallets $\Phi 4$, $\Phi 2$, $\Phi 3$ does not have visible effects but is a prerequisite for the second iteration to happen and be succesful. The effects of the second iteration start with the $\Phi 4.L4$ execution that can determine the words of W_{before} .

3.4. Pattern P3: Input Word Capture

Situation : $W_i \neq \emptyset$

The situation refers to the fact that the transition must capture input words. Therefore, this situation shall occur in any SF execution implementing a complete interaction with the user, as the assumption is that the transition includes at least one capture of input words from the external world.

Execution trace :

Step	Trace	Effect
(1)	$\Phi 2.\{L2,i\}$, i : input words	None
(2)	$\Phi 2.I2$	Read the input from physical device
(3)	Loop	NextPallet Id=SF01W02
(4)	$\Phi 2.\{L2,i\}$, i : input words	Determine the input words (from buffer)
(5)	$\Phi 2.I2$	None
(6)	$\Phi 2.R2$	NextPallet Id=SF01W03
(7)	$\Phi 3.\{L3,j\}$, j : output word	None
(8)	$\Phi 3.R3r$ (or $\Phi 3.R3c$, c : continuous link)	NextPallet Id = SF01W04

Table 4 : Execution Trace of Pattern P3

The trace presented in Table 4 indicates that two iterations of the pallet function $\Phi 2$ are needed to successfully capture the input words. The first iteration makes the physical *read action* feasible whereas the second uses the buffer filled in by the *read action* to actually determine the input words. The control is then given to pallet $\Phi 3$.

3.5. Pattern P4: Output Word Production

Situation : $W_{\text{after}} = f(W_{\text{input}})$

This situation indicates that W_{after} depends of W_{input} . This situation shall occur in any SF execution implementing a complete interaction with the user (or interfacing a program with a database).

Execution trace :

Step	Trace	Effect
(1)	$\Phi 3.\{L3,j\}, j$: output words	Determine the necessary conditions of output words
(2)	$\Phi 3.R3r$ (r : recursive link)	NextPallet Id=SF01W04
(3)	$\Phi 4.\{L4,j\}, j$: output words	Determine the output words dependent of the input words
(4)	$\Phi 4.O4$	Display the screen / Write in the database
(5)	$\Phi 4.S4$	Clear word memory / None
(6)	$\Phi 4.R4$	NextPallet Id=SF01W02 Wait for the user interaction (asynchronous relation between pallets W04 and W02) Or No Wait (synchronous relation between pallets W04 and W02)

Table 5 : Execution Trace of Pattern P4

As shown in Table 5, the pattern indicates that in the production of output words, the execution of the pallet function $\Phi 3$ determining the necessary conditions of the output words is followed by the execution of the pallet function $\Phi 4$ implementing physically the write operation on the corresponding device.

3.6. Pattern P5: Dependent Output Word Production

Situation : $\exists j1 \in W_{\text{after}}$ and $\{j: j \in W_{\text{after}} \text{ and } j = f_j(j1)\} \neq \emptyset$

The characteristic of this situation is that some of the output words produced in the SF are intermediate values which are not displayed to the user.

This situation addresses dependencies among output words belonging to W_{after} . More precisely the situation identifies a dependence of the type *n to 1* : n output words $\{j\}$ belonging to W_{after} are dependent of only 1 word $j1$ in the same W_{after} .

This situation imposes an ordering in the calculation of output words, which is reflected in the pattern sequence of steps.

Execution trace :

Step	Trace	Effect
(1)	$\Phi 3.\{L3,j\}$ j : output words	determine the necessary conditions of output words j1 and {j}
(2)	$\Phi 3.R3r$ r : recursive link	NextPallet Id=SF01W04
(3)	$\Phi 4.\{L4,j1\}$ j1 : output words	Determine the output word j1 dependent of the input words
(4)	$\Phi 4.O4$	None
(5)	Loop	NextPallet Id=SF01W04
(6)	$\Phi 4.\{L4,j\}$ {j} : dependent output words	Determine the output words {j}
(7)	$\Phi 4.O4$	Display the screen / Write in the database
(8)	$\Phi 4.S4$	Clear word memory / None
(9)	$\Phi 4.R4$	NextPallet Id=SF01W02 Wait for the user interaction (asynchronous relation between pallets W04 and W02) Or No Wait (synchronous relation between pallets W04 and W02)

Table 6 : Execution Trace of Pattern P5

The key characteristic of a sequence of steps presented in Table 6 is a double execution of the $\Phi 4$ pallet function : one for determining the dependee word ($j1$) and one for determining the set of dependent words. It shall be noticed that two iterations are sufficient, even if the set of dependent words $\{j\}$ comprises more than one word.

3.7. Pattern P6 : Termination of a SF Execution

Situation: W_{after} is determined

The situation refers to the fact that the word W_{after} have already been evaluated.

Execution trace :

Step	Trace	Effect
(1)	$\Phi 4.\{L4,j\}$ j : set of output words	Already determined
(2)	$\Phi 4.O4$	None
(3)	$\Phi 4.S4$	None
(4)	$\Phi 4.R4$	NextPallet Id=SF01W02
(5)	$\Phi 2.\{L2,i\}$ i : set of input words	Already determined
(6)	$\Phi 2.I2$	None
(7)	$\Phi 2.R2$	NextPallet Id=SF01W03
(8)	$\Phi 3.\{L3,j\}$ j : set of output words	Already satisfied
(9)	$\Phi 3.\{R3,k\}$ k : continuous, duplex or multiplex	→End of the Scenario Function SF01.

Table 7 : Execution Trace of Pattern P6

As shown in table 7, a SF termination is detected when one iteration of the three pallet functions induces no change compared to the previous iteration. Consequently the pattern trace is made of a series of steps reflecting that the three pallet function executions have no effect.

3.8. Pattern P7 : Mono Interaction

Situation : W_i = Capture_{one shot} (W_{i1})

As shown in Figure 4 the situation of this pattern refers to a transition in which a single set of input words is captured in one shot. Such an SF is called atomic.

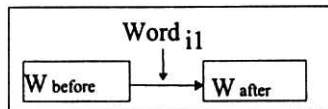


Figure 4 : Single Interaction

Execution Trace = $(P1 \mid P2) + [P3] + [P4 + (P5)*] + P6$

In this formula, the name of a pattern is used instead of the complete execution trace corresponding to it. The vertical bar expresses that the left item and right item can be substituted one to the other. A bracket means that the item is optional. Finally, the plus operator expresses the trace concatenation. Pattern P7 is a compound pattern which provides the composition of patterns in a transition with one single interaction.

Pattern P1 or Pattern P2 are applied to Start the execution of the transition; the pattern P3 is applied to capture the set of input words W_{ij} . The pattern P4 or the pattern P5 (several times) are then be applied in order to calculate the output words; depending of the fact that they are dependent, or not of the others words. Pattern P6 ends the transition.

3.9. Pattern P8 : Multi – Interaction

Situation : $W_i = \sum_j \text{Capture}_{\text{one shot}} (W_{ij})$

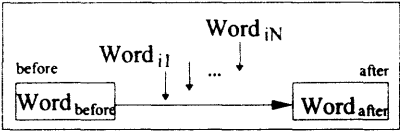


Figure 5: Multi-Interaction

The situation refers to in this pattern is the one of a *compound* SF. The term compound is employed here to express the fact that within the same SF, several interactions are performed to capture the input words (Figure 5).

Execution Trace = $(P1 \mid P2) + ([P3] + [P4 + (P5)^*])^N + P6$

This pattern is similar to P7 but corresponds to the execution a *compound* SF. The sequence of steps it refers to is an aggregate of other pattern sequences of steps .

4. Applying Patterns

In order to illustrate the *execution patterns*, we present the *Split a Goal* example. Some other examples can be found in [8]. *Split a Goal* is a functionality which, given a goal statement such as ‘Withdraw cash from an ATM’, automatically decomposes it into a *verb* and its *parameters*. For example, *Withdraw* is the *verb*, *cash* is the *target* parameter of the verb and *from an ATM* is the *means* parameter.

The full functionality identifies 7 different parameters. However, in this paper we will consider only the two parameters exemplified above, *target* and *means*.

The *Split a Goal* example includes two screens: the *Input screen* (Screen1) lets the user input the goal and inquiry for its decomposition (Figure 6). The *Output screen* (Screen2) displays the values of the parameters resulting of the decomposition of the goal (Figure 7). When the user clicks on the *SearchDB* button, the system accesses to the database using the *GoalId* as query parameter and retrieves the goal name which is displayed to the user in the *Goal* field.

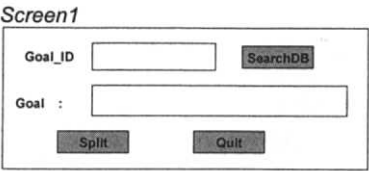


Figure 6: The *Input screen* to formulate the goal

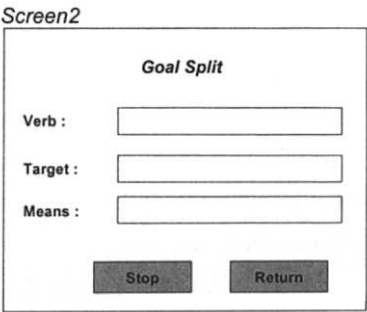


Figure 7: The *Output screen* to display the parameters

When the user clicks on the *Split* button, the system decomposes the goal into a *Verb* and its parameters (*Target*, *Means*) using an algorithm, and displays the results on the *Output screen* (Figure 7). The *Return* button triggers the display of the input screen whereas the *Stop* button allows the user to stop the process at any moment.

The PRD presented in Figure 8 is composed of three Scenario Functions namely SF01 (associated to Screen 1), SF02 (for accessing the database) and SF03 (for displaying outputs in Screen 2):

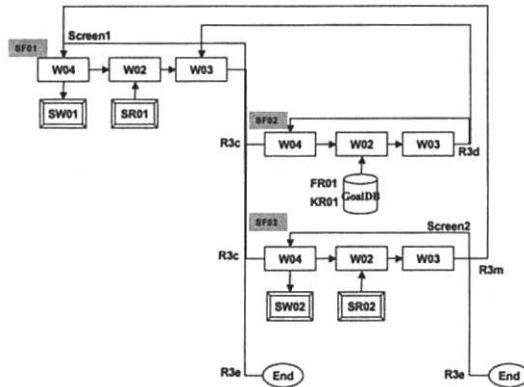


Figure 8: Process Route Diagram for *Split a Goal* example

Table 8 presents the execution trace, column 2 relates a sequence of steps in the trace to one of the 8 patterns.

Step	Pattern Applied	Trace	Effect
(1)	P8	Φ4 L4.Goal	None
(2)		O4	Display screen
(3)		S4	Clear memory word
(4)		R4	NextPallet Id = SF01W02
		<i>Wait until the user action</i>	<i>The user clicks on SearchDB button</i>
(5) (6)		Φ2 L2.Goal_ID, L2.SearchDB	None
(7) (8)		L2.Split, L2.Quit	None
(9)		I2	Capture input from screen
(10)		<i>Loop</i>	NextPallet Id = SF01W02
(11) (12)		Φ2 L2.Goal_ID, L2.SearchDB	Determine Input word (from buffer)
(13) (14)		L2.Split, L2.Quit	Determine Input word (from buffer)
(15)		I2	None
(16)		R2	NextPallet Id = SF01W03
(17)		Φ3 L3.Goal	Determine the necessary condition
(18)		R3c (continous link)	NextPallet Id = SF02W04 → end of execution of SF01 (break)
(19)	P7	Φ4 L4.Key_GoalID	None
(20)		O4	None
(21)		S4	None
(22)		R4	NextPallet Id = SF02W02
(23)		Φ2 L2.Goaldb	None
(24)		I2	None
(25)		R2	NextPallet Id = SF02W03
(26)		Φ3 L3.Key_GoalID	Determine the necessary condition
(27)		R3r	NextPallet Id = SF02W04
(28)		Φ4 L4.Key_GoalID	Determine Output word
(29)		O4	None
(30)		S4	None
(31)		R4	NextPallet Id = SF02W02
(32)		Φ2 L2.Goaldb	None
(33)		I2	Capture input from database
(34)		<i>Loop</i>	NextPallet Id = SF02W02

(35)	P6	$\Phi 2$ L2.Goaldb I2 R2	Determine input word from buffer None NextPallet Id = SF02W03
(36)		$\Phi 3$ L3.Key_GoalID R3r	Already Satisfied NextPallet Id = SF02W04
(37)			
(38)			
(39)			
(40)			
(41)	P8	$\Phi 4$ L4.Key_GoalID O4 S4 R4	Already determined None None NextPallet Id = SF02W02
(42)		$\Phi 2$ L2.Goaldb I2 R2	Already determined None NextPallet Id = SF02W03
(43)		$\Phi 3$ L3.Key_GoalID R3d (<i>duplex link</i>)	Already satisfied NextPallet Id = SF01W03 → end of execution of SF02
(44)			
(45)			
(46)			
(47)	P3	$\Phi 3$ L3.Goal R3r (<i>return of the duplex link</i>)	Already determined NextPallet Id = SF01W04
(48)		$\Phi 4$ L4.Goal O4 S4 R4 <i>Wait until the user action</i>	Determine output word Display screen Clear memory word NextPallet Id = SF01W02 The user clicks on the Split button
(49)		$\Phi 2$ L2.Goal_ID, L2.SearchDB L2.Split, L2.Quit I2 <i>Loop</i>	None None Capture input from screen NextPallet Id = SF01W02
(50)		$\Phi 2$ L2.Goal_ID, L2.SearchDB L2.Split, L2.Quit I2 R2	Determine Input word (from buffer) Determine Input word (from buffer) None NextPallet Id = SF01W03
(51)		$\Phi 3$ L3.Goal R3r	Already satisfied NextPallet Id = SF01W04
(52)			
(53)	P6	$\Phi 4$ L4.Goal O4 S4 R4	Already determined None None NextPallet ID = SF01W02
(54)		$\Phi 2$ L2.Goal_ID, L2.SearchDB L2.Split, L2.Quit I2 R2	Already determined Already determined None NextPallet Id = SF01W03
(55)		$\Phi 3$ L3.Goal R3c (<i>continous link</i>)	Already satisfied NextPallet Id = SF03W04 → end of execution of SF01
(56)			
(57)			
(58)			
(59)	P2	$\Phi 4$ L4.Verb, L4.Target, L4.Mean O4 S4 R4	None None None NextPallet Id = SF03W02
(60)		$\Phi 2$ L2.Stop, L2.Return I2 R2	None None NextPallet Id = SF03W03
(61)		$\Phi 3$ L3.Verb, L3.Target, L3.Mean R3r	Determine the necessary condition NextPallet Id = SF03W04
(62)		$\Phi 4$ L4.Verb, L4.Target, L4.Mean O4 S4 R4 <i>Wait until the user action</i>	Determine output word Display screen Clear memory word NextPallet Id = SF03W02 The user clicks on the Return button
(63)		$\Phi 2$ L2.Stop, L2.Return I2 <i>Loop</i>	None Read input from input screen NextPallet Id = SF03W02
(64)		$\Phi 2$ L2.Stop, L2.Return I2 R2	Determine input word None NextPallet Id = SF03W03
(65)	P3	$\Phi 3$ L3.Verb, L3.Target, L3.Mean R3r	Already satisfied NextPallet Id = SF03W04
(66)			
(67)			
(68)			
(69)			
(70)			
(71)	P6	$\Phi 4$ L4.Verb, L4.Target, L4.Mean O4 S4 R4	Already determined None None NextPallet Id = SF03W02
(72)		$\Phi 2$ L2.Stop, L2.Return I2 R2	Determine input word None NextPallet Id = SF03W03
(73)		$\Phi 3$ L3.Verb, L3.Target, L3.Mean R3r	Already satisfied NextPallet Id = SF03W04
(74)			
(75)			
(76)			
(77)	P2	$\Phi 4$ L4.Verb, L4.Target, L4.Mean O4 S4 R4	Already determined None None NextPallet Id = SF03W02
(78)		$\Phi 2$ L2.Stop, L2.Return I2 R2	Determine input word None NextPallet Id = SF03W03
(79)		$\Phi 3$ L3.Verb, L3.Target, L3.Mean R3r	Already satisfied NextPallet Id = SF03W04
(80)			
(81)			
(82)			
(83)	P3	$\Phi 4$ L4.Verb, L4.Target, L4.Mean O4 S4 R4	Already determined None None NextPallet Id = SF03W02
(84)		$\Phi 2$ L2.Stop, L2.Return I2 R2	Determine input word None NextPallet Id = SF03W03
(85)		$\Phi 3$ L3.Verb, L3.Target, L3.Mean R3r	Already satisfied NextPallet Id = SF03W04
(86)			
(87)			
(88)			
(89)	P6	$\Phi 4$ L4.Verb, L4.Target, L4.Mean O4 S4 R4	Already determined None None NextPallet Id = SF03W02
(90)		$\Phi 2$ L2.Stop, L2.Return I2 R2	Determine input word None NextPallet Id = SF03W03
(91)		$\Phi 3$ L3.Verb, L3.Target, L3.Mean R3r	Already satisfied NextPallet Id = SF03W04
(92)			
(93)			
(94)			
(95)	P2	$\Phi 4$ L4.Verb, L4.Target, L4.Mean O4 S4 R4	Already determined None None NextPallet Id = SF03W02
(96)		$\Phi 2$ L2.Stop, L2.Return I2 R2	Determine input word None NextPallet Id = SF03W03
(97)		$\Phi 3$ L3.Verb, L3.Target, L3.Mean R3r	Already satisfied NextPallet Id = SF03W04
(98)			
(99)			
(100)			
(101)	P3	$\Phi 4$ L4.Verb, L4.Target, L4.Mean O4 S4 R4	Already determined None None NextPallet Id = SF03W02
(102)		$\Phi 2$ L2.Stop, L2.Return I2 R2	Determine input word None NextPallet Id = SF03W03
(103)		$\Phi 3$ L3.Verb, L3.Target, L3.Mean R3r	Already satisfied NextPallet Id = SF03W04
(104)			
(105)			
(106)			
(107)	P6	$\Phi 4$ L4.Verb, L4.Target, L4.Mean O4 S4 R4	Already determined None None NextPallet Id = SF03W02
(108)		$\Phi 2$ L2.Stop, L2.Return I2 R2	Determine input word None NextPallet Id = SF03W03
(109)		$\Phi 3$ L3.Verb, L3.Target, L3.Mean R3r	Already satisfied NextPallet Id = SF03W04
(110)			
(111)			
(112)			
(113)	P2	$\Phi 4$ L4.Verb, L4.Target, L4.Mean O4 S4 R4	Already determined None None NextPallet Id = SF03W02
(114)		$\Phi 2$ L2.Stop, L2.Return I2 R2	Determine input word None NextPallet Id = SF03W03
(115)		$\Phi 3$ L3.Verb, L3.Target, L3.Mean R3r	Already satisfied NextPallet Id = SF03W04
(116)			
(117)			
(118)			

(118) (119) (120) (121) (122) (123) (124) (125)			Φ2 L2.Stop, L2.Return I2 R2	Already determined None NextPallet Id = SF03W03
			Φ3 L3.Verb, L3.Target, L3.Mean R3m (<i>multiplex link</i>)	Already satisfied NextPallet Id = SF01W04 → <i>end of execution of SF03</i>
(126) (127) (128) (129) (130) (131) (132) (133) (134) (135) (136) (137) (138) (139) (140) (141) (142) (143)	P7	P1	Φ4 L4.Goal O4 S4 R4 <i>Wait until the user action</i>	None Display screen Clear memory word NextPallet Id = SF01W02 <i>The user clicks on the Quit button</i>
		P3	Φ2 L2.Goal_ID, L2.SearchDB L2.Split, L2.Quit I2 <i>Loop</i>	None None Capture input from screen NextPallet Id = SF01W02
			Φ2 L2.Goal_ID, L2.SearchDB L2.Split, L2.Quit I2 R2	Determine Input word (from buffer) Determine Input word (from buffer) None NextPallet Id = SF01W03
			Φ3 L3.Goal R3r	None NextPallet Id = SF01W04
			P6	Φ4 L4.Goal O4 S4 R4
		Φ2 L2.Goal_ID, L2.SearchDB L2.Split, L2.Quit I2 R2		Already determined Already determined None NextPallet Id = SF01W03
(144) (145) (146) (147)(148) (149) (150) (151) (152) (153) (154) (155)	Φ3 L3.Goal R3e (continuous link)	Already satisfied NextPallet Id = END → <i>end of execution of program</i>		

Table 8: Relating *Split a Goal* Trace to Patterns

Table 8 shows that the trace is composed of 14 parts each corresponding to a sequence of steps of one of the six atomic patterns P1 to P6.

In addition, the table shows that groups of sequences of steps conform to the compound patterns P7 and P8. For example, it can be seen that the trace starts with an instance of the P8 pattern, followed by two instances of pattern P7.

The instance of pattern P8 conforms the P8 template : it is composed of a sequence of atomic patterns instances : P1, P3, P4, P3 and P6.

5. Conclusion

This paper reflects on our experience with LyeeALL. It proposes generic Lyee program execution traces captured in *Lyee execution patterns*. Each pattern identifies a recurrent sequence of steps in Lyee program execution control and to relate it to the situation in which this sequence shall be executed.

Any Lyee program execution trace results of the assembly of pattern sequences. In other words, any Lyee program execution trace is an aggregation of instances of some of these execution patterns. This was shown in the *Split a Goal* example. The execution patterns were useful to understand the relevant design situations when drawing PRDs. Based on these situations a set of design patterns have been defined and are currently under validation.

6. References

- [1] C. Alexander, S. Ishikawa, M. Silverstein, M. Jacobson, I. Fiksdahl-King, S. Angel, "*A Pattern Language*", Oxford University Press, New York, 1977.
- [2] J. Coplien, D. Schmidt (eds.), "*Pattern Languages of Program Design*", Addison Wesley, Reading, MA, 1995.
- [3] M. Fowler, "*Analysis Patterns: Reusable Object Models*", Addison-Wesley, 1997.
- [4] E. Gamma, R. Helm, R. Johnson, J. Vlissides, "*Design Patterns: Elements of Reusable Object-Oriented Software*", Addison Wesley, Reading, MA, 1995.
- [5] F. Negoro, "*Intent Operationalisation For Source Code Generation*", Proceedings of SCI and ISAS, Orlando, 2001.
- [6] F. Negoro, "*Methodology to define software in a deterministic manner*", Proceedings of ICII, Beijing, China, 2001.
- [7] F. Negoro, "*The predicate structure to represent the intention for software*", Proceedings of SNPD, Nagoya, Japan, 2001.
- [8] S. Nurcan, M. Ben Ayed, C. Rolland, Lyee International Collaborative Project University Paris 1 Scientific Report SC1, Mars 2002
- [9] C. Rolland, M. Ben Ayed, "*Understanding the Lyee Methodology through Meta Modelling*", Proceeding of EMMSAD, Toronto, 2002.

Invited Presentation 3

This page intentionally left blank

Bringing HCI Research to Bear Upon End-User Requirement Specification

Margaret BURNETT

Department of Computer Science, Oregon State University, Corvallis, OR 97331 USA

Abstract. Can end users someday enter their requirements directly into program generation tools? This paper surveys literature that sheds some light upon this question, including devices from HCI research that can be used to help in accomplishing this goal.

1. Introduction

One goal envisioned by the inventors of the Lyee methodology [12] that is the subject of this workshop is that even an end user will someday be able to enter a set of software requirements—without the assistance of a professional software developer—and that the methodology will be able to generate software that conforms to these requirements. (Following convention in the literature, the term *end user* as used here refers to people who are not professional programmers.) Indeed, other collaborators at this workshop have been working on important fundamentals that will contribute to this goal (e.g., [17, 18]). Still, in order for this goal to become a reality, many questions relating to the human aspects of this goal must be investigated, such as:

- Is it feasible to expect end users to enter software requirements at all?
- Even if so, to what level of detail must they descend in order to create software requirements that are complete enough to generate the desired software?
- How can we ensure that requirements entered by such users are non-contradictory?
- Can end users understand the implications of the requirements they are entering well enough to recognize an error and correct it?

There is relevant research contributing partial answers to the above questions. These partial results allow cautious optimism that the questions above can be resolved. In this paper, we present a brief survey of some of this relevant research, and also discuss some of the underlying principles that they help to demonstrate. Unlike many of the other papers at this workshop, the purpose of this paper is not to report progress on a particular project, but rather to present tutorial material on principles and strategies to support end-user requirement specification.

2. End-User Requirement Specification

Requirement specification can be either supplemental information to a program, as in specifying assertions about various portions of a program, or can be a complete specification of a program. We focus in this section on the former.

Research indicates that end users can potentially work with some forms of requirement specifications. Nardi summarized work by several researchers indicating that, although end users are not particularly good at working with abstract requirements, end users work much better with a concrete program they are able to criticize [11]. However, there has been only a little research following up on these findings by making requirement specification mechanisms available to end users.

Some of our research relates to exactly this problem. We briefly summarize it here, to provide a concrete example of supporting end users' requirement specifications.

In general, we have been working on how to improve the reliability of end-user programs in general and spreadsheets in particular [13, 14, 15, 16]. One of our hypotheses is that spreadsheet reliability can be improved if the spreadsheet users work collaboratively with the system to communicate additional information about known relationships. In other words, spreadsheet users know more about the purpose and underlying requirements for their spreadsheets than they are currently able to communicate to the system, and our goal is to allow end users to communicate this information about requirements. This will allow checks and balances, so that the system can detect and point out ways in which the spreadsheet does not conform to the user's requirements.

We are pursuing the question of requirement specifications for end users [1, 22] using the research spreadsheet language Forms/3 [3]. In our prototype, we refer to requirement specifications as *guards*. Guards are the same concept as assertions. We began this work with an early prototype for individual cells in spreadsheets, which afforded empirical investigation into how users problem solve in the presence of guards [22].

In our earliest prototype (shown in Figure 1), guards pertained to only one cell [22]; we have since extended the prototype to allow guards to pertain to multiple cells [1]. In both cases, a guard's implications are propagated through formulas to other cells. That is, whenever a user puts a guard into a spreadsheet (a *user-entered guard*) it is propagated through formulas downstream generating *computer-generated guards* on downstream cells. A cell with both a computer-generated and user-entered guard is in a conflict state (a *guard conflict*) if the two guards do not match exactly. As Figure 1 shows, to communicate the fact that there is a guard conflict, the system circles the conflicting guards. The stick figure icon shown in Figure 1 identifies a guard that was explicitly entered by the user. The computer icon identifies a computer-generated guard on cell Celsius, which the system generated by propagating the Fahrenheit guard through Celsius's formula. Because it does not match the user-entered guard on Celsius, there is a guard conflict, indicated by the circle around the two guards. Since the cell's value is inconsistent with a guard on that cell (termed a *value violation*), the value is also circled.

Using both our prototype and paper-and-pencil sketches, we have conducted studies to see whether end users can use assertions effectively [1, 22]. Some of our findings are that



Figure 1: A Forms/3 temperature converter. The stick figure icon identifies a user-entered guard, and the computer icon identifies a computer-generated guard. The computer-generated guard's conclusion that the Celsius value ranges from 0 to 324 degrees provides a clue that there is an error in Celsius's formula.

end users can indeed work with assertions, that the assertions brought errors to their attention, and that the users found the cross-checking provided by the assertions reassuring. Further, all users with guards were able to understand guard propagation when it occurred in a live system (but not all were as successful without the live system's presence), and were usually not drawn off task by value violations and/or guard conflicts. The guard subjects who made errors were directly influenced by the guards in finding and correcting the errors. Finally, the guards did not cause users to do less testing.

3. End-User Programming and its Importance to End-User Requirement Specification

A complete specification of a program is itself a (declarative) program. From this it is clear that supporting end-user requirement specifications that are complete is the same as supporting end-user programming. Hence, the body of research on end-user programming languages is highly relevant to the question of end-user specification.

Is it possible for end users to create programs? Do they even want to? There is abundant evidence that the answer to both questions is yes. In fact, spreadsheet end users have been "programming" for many years: every time they enter or edit a cell's formula, they are engaging in first-order functional programming.

However, traditional programming languages are not what end-user programmers use—even when such languages have been cosmetically altered with visual front-ends. Instead, they use "programming languages" that have been designed from the ground up for end users. Examples include spreadsheet systems, web authoring and multimedia authoring tools, simulation builders, and visualization tools. Although none of these devices are marketed as "programming languages," they fulfill the attributes normally expected of programming languages, supporting sequence, selection (conditionals), and condition-based repetition. Some of them (e.g., Cocoa [6]) are even Turing-complete.

There are four common strategies used in designing end-user programming languages. These strategies cannot be viewed as a "how to succeed" plan in creating end-user programming languages, but they do help demonstrate some common differences between end-user programming languages and traditional languages. The common strategies are:

Concreteness: Concreteness is the opposite of abstractness, and means expressing some aspect of a program using particular instances. One example is allowing a user to specify some aspect of semantics on a specific object or value, and another example is having the system automatically display the effects of some portion of a program on a specific object or value.

Directness: Directness in the context of direct manipulation is usually described as "the feeling that one is directly manipulating the object" [19]. From a cognitive perspective, directness in computing means a small distance between a goal and the actions required of the user to achieve the goal [5, 7, 11]. Given concreteness in a language, an example of directness would be allowing the user to manipulate a specific object or value directly to specify semantics rather than describing these semantics textually. For example, if the user wants an object to move near the top of the screen at a quick rate a speed, a direct way of expressing this logic is to move it at the desired speed to the desired location, as compared to a traditional, much less direct, expression of these semantics such as "move object4 to: (14,3) rate: 23" (or, more tersely, "object4.move(14,3,23)").

Explicitness: Some aspect of semantics is explicit in the environment if it is directly

stated (textually or visually), without the requirement that the user infer it. An example of explicitness in a language would be for the system to explicitly depict dataflow relationships by drawing directed edges among related objects.

Immediate Visual Feedback: In the context of programming, immediate visual feedback refers to automatic display of effects of program edits. Tanimoto has coined the term *liveness*, which categorizes the immediacy of semantic feedback that is automatically provided during the process of editing a program [21]. Tanimoto described four levels of liveness. The highest two (levels 3 and 4) are of most interest in end-user programming, because it is at level 3 where semantic feedback begins. More specifically, at level 3 incremental semantic feedback is automatically provided whenever the programmer performs an incremental program edit, and all affected onscreen values are automatically redisplayed. This ensures the consistency of display state and system state if the only trigger for system state changes is programmer editing. The automatic recalculation feature of spreadsheets supports level 3 liveness. At level 4, the system responds to program edits as in level 3, and to other events as well such as system clock ticks and mouse clicks over time, ensuring that all data on display accurately reflects the current state of the system as computations continue to evolve.

Figure 2, Figure 3, Figure 4, and Figure 5 show examples of end-user programming languages.

Concreteness is quite evident in these figures. Directness also plays an important role: the user directly puts the object into the desired state. Explicitness and immediate feedback are supported in varying degrees in these languages, but are not apparent in most of these screen shots. Most end-user programming languages to date have been domain-specific languages. The reason for this tendency is easy to see, given the common strategy of directness; a domain-specific vocabulary (graphical or textual) of language primitives allows users to express semantics in a vocabulary very close to goals they are trying to accomplish. The domain of Chimera [8] (Figure 2) is graphical editing. The domain of Cocoa [20] (Figure 3) is graphical simulations and games, although it has been demonstrated to be very effective at robot programming also. (Cocoa, formerly known as KidSim and now known commercially as Stagecast Creator, is a Turing-complete language.) The domain of FAR [4] (Figure 4) is web programming. The domain of SILK [9] (Figure 5) is user interface specification.

SILK (Figure 5) is particularly applicable to the Lyee methodology, because Lyee requirements rest upon screen diagrams. A possible relationship between Lyee and the SILK approach, which could also draw from the formulas and rules of FAR (Figure 4) or rules of Cocoa (Figure 3), is discussed further in the companion paper [2].

4. Underlying Principles

Since the goals of end-user programming and end-user software engineering devices have to do with empowering humans in their problem solving endeavors, it is important for designers of such devices to consider what is known about cognitive issues relevant to programming. If consideration of these issues is incorporated into the design from the very outset, it becomes possible to detect—early in the design cycle—design decisions that are at odds with human problem solving needs.

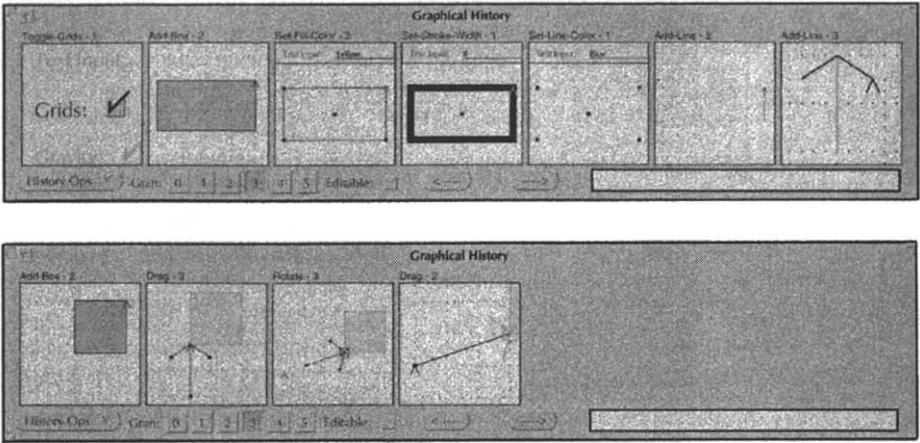


Figure 2: Programming by demonstration in Chimera. In this example, the user has drawn a box with an arrow pointing to it (as in a graph diagram), and this demonstration is depicted after-the-fact in the series of intelligently-filtered panels shown, following a comic strip metaphor. The user can tell the system to generalize this set of demonstrations into a macro for use in creating the other nodes in the graph semi-automatically.

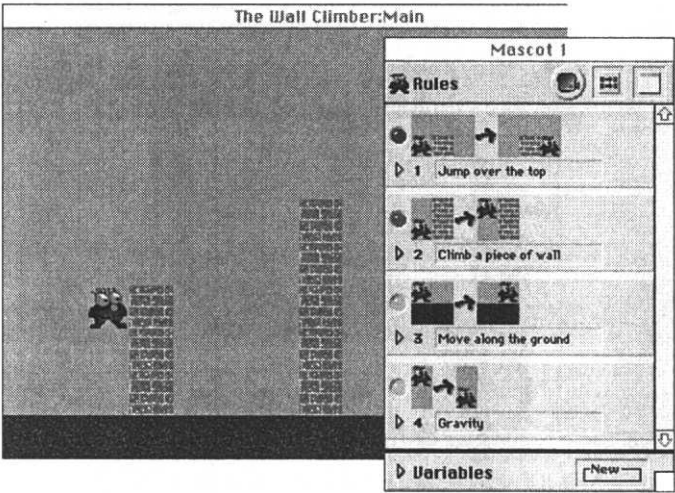


Figure 3: A Cocoa wall-climber (The Wall Climber: Main window) is following the rules (Mascot 1 window) that have been demonstrated for it. Each rule is shown with the graphical precondition on the left of the arrow and the graphical postcondition on the right of the arrow. The wall climber has just finished following rule 2, which places it in a position suitable for following rule 1 next.

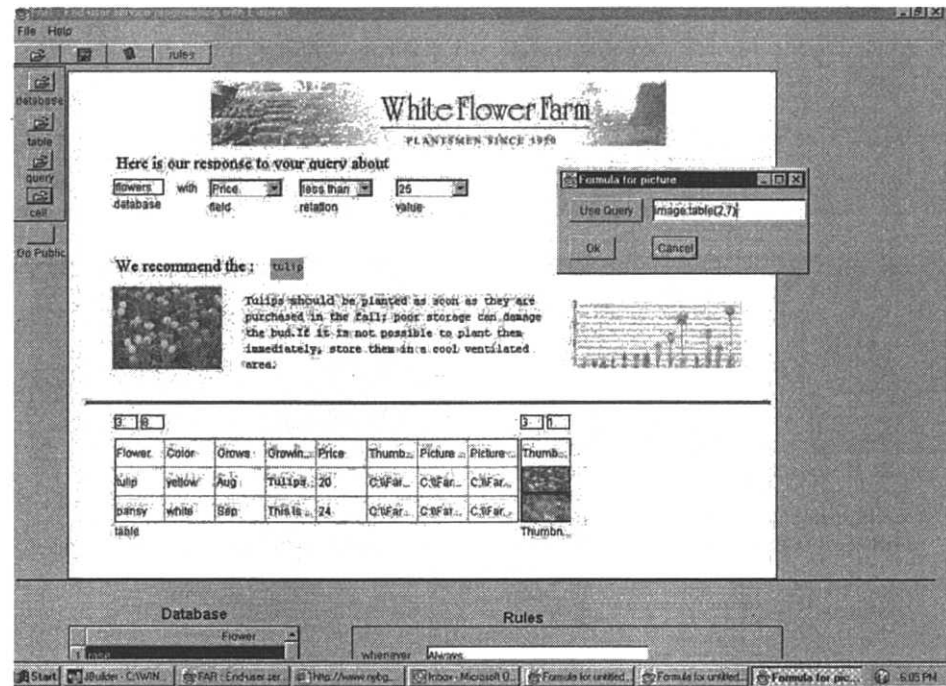


Figure 4: Snapshot of the flower advisor e-service in FAR. The user places objects as desired on the web page area, and specifies formulas that can depend on external databases and incoming query information. Everything shown in the web page area is a cell or table of cells, including the blocks of text, images, and so on. The formula of the cell showing a field of tulips (mid left) is shown in the inset window at right.

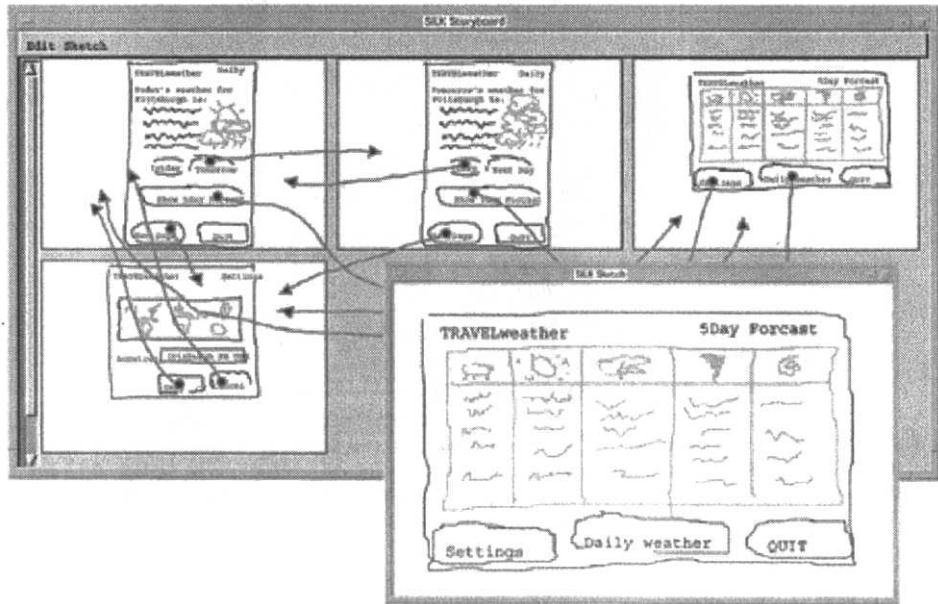


Figure 5: A SILK sketch (front) of a five-day weather forecast and storyboard (rear) [9]. An experienced user-interface designer created the sketch. The designer has also created buttons and drawn arrows to show the screen transitions that occur when the user presses the buttons.

The most influential research project to date that converts cognitive and human-computer interactions findings into a form that can be used by designers in the process of designing end-user programming systems is Cognitive Dimensions [5]. Cognitive Dimensions (CDs) are a set of terms describing the structure of a programming language's components as they relate to cognitive issues in programming.

Table 1 lists the dimensions, along with a thumb-nail description of each. The relation of each dimension to a number of empirical studies and psychological principles is given in [5], but the authors also carefully point out the gaps in this body of underlying evidence. In their words, "The framework of cognitive dimensions consists of a small number of terms which have been chosen to be easy for non-specialists to comprehend, while yet capturing a significant amount of the psychology and HCI of programming."

CDs help to clarify principles behind the strategies covered in the previous section. For example, directness in a language's vocabulary is the same concept as the closeness of mapping CD. Immediate visual feedback is an example of progressive evaluation. In order to accomplish immediate visual feedback, concrete sample values are usually required. Concreteness is also related to the abstraction gradient CD above. Explicitness includes eliminating the hiddenness of dependencies.

CDs have been used by a number of researchers to evaluate the cognitive aspects of visual programming languages and end-user programming languages, and to make broad comparisons of cognitive aspects of such languages. For example, Green and Petre used CDs to contrast cognitive aspects of the commercial visual programming languages Prograph and LabVIEW [5]. Modugno also used CDs to evaluate Pursuit, a research programming-by-demonstration language [10]. The real strength of CDs though is as a design-time device, to head off design problems at an early stage. Many researchers have turned to CDs for this purpose; one example of a language designed with the help of CDs from the ground up is FAR [4] (Figure 4).

Table 1: The cognitive dimensions.

Abstraction gradient	What are the minimum and maximum levels of abstraction? Can fragments be encapsulated?
Closeness of mapping	What 'programming games' need to be learned? That is, can the user express problem solutions directly, without having to learn circuitous ways to cause the goal to be accomplished?
Consistency	When some of the language has been learnt, how much of the rest can be inferred?
Diffuseness	How many symbols or graphic entities are required to express a meaning?
Error-proneness	Does the design of the notation induce 'careless mistakes'?
Hard mental operations	Are there places where the user needs to resort to fingers or penciled annotation to keep track of what's happening?
Hidden dependencies	Is every dependency overtly indicated in both directions? Is the indication perceptual or only symbolic?
Premature commitment	Do programmers have to make decisions before they have the information they need?
Progressive evaluation	Can a partially-complete program be executed to obtain feedback on "How am I doing"?
Role-expressiveness	Can the reader see how each component of a program relates to the whole?
Secondary notation	Can programmers use layout, color, or other cues to convey extra meaning, above and beyond the 'official' semantics of the language?
Viscosity	How much effort is required to perform a single change?
Visibility	Is every part of the code simultaneously visible (assuming a large enough display), or is it at least possible to compare any two parts side-by-side at will? If the code is dispersed, is it at least possible to know in what order to read it?

Table 2: Summary of the representation design benchmarks. S_c denotes measures of the characteristics of elements of S . S_D denotes measures of the presence of potential elements of S . Each S_D measure has a corresponding NI measure.

Benchmark Name	S_c	S_D	NI	Aspect of the Representation	Computation
D1		x		Visibility of dependencies	(Sources of dependencies explicitly depicted) / (Sources of dependencies in system)
D2			x		The worst case number of steps required to navigate to the display of dependency information
PS1		x		Visibility of program structure	Does the representation explicitly show how the parts of the program logically fit together? Yes/No
PS2			x		The worst case number of steps required to navigate to the display of the program structure
L1		x		Visibility of program logic	Does the representation explicitly show how an element is computed? Yes/No
L2			x		The worst case number of steps required to make all the program logic visible
L3	x				The number of sources of misrepresentations of generality
R1		x		Display of results with program logic	Is it possible to see results displayed statically with the program source code? Yes/No
R2			x		The worst case number of steps required to display the results with the source code.
SN1		x		Secondary notation: non-semantic devices	$SN_{devices} / 4$ where $SN_{devices}$ = the number of the following secondary notational devices that are available: optional naming, layout devices with no semantic impact, textual annotations and comments, and static graphical annotations.
SN2			x		The worst case number of steps to access secondary notations
AG1		x		Abstraction gradient	$AG_{sources} / 4$ where $AG_{sources}$ = the number of the following sources of details that can be abstracted away: data details, operation details, details of other fine-grained portions of the programs, and details of NI devices.
AG2			x		The worst case number of steps to abstract away the details
RI1		x		Accessibility of related information	Is it possible to display all related information side by side? Yes/No
RI2			x		The worst case number of steps required to navigate to the display of related information.
SRE1	x			Use of screen real estate	The maximum number of program elements that can be displayed on a physical screen.
SRE2	x				The number of non-semantic intersections on the physical screen present when obtaining the SRE1 score
AS1, AS2, AS3	x x x			Closeness to a specific audience's background	$AS_{yes's} / AS_{questions}$ where $AS_{yes's}$ = the number of "yes" answers, and $AS_{questions}$ = the number of itemized questions of the general form: "Does the <representation element> look like the <object/operation/ composition mechanism> in the intended audience's prerequisite background?"

A concrete application of the Cognitive Dimensions is *representation design benchmarks* [23], a set of quantifiable measurements that can be made on a language environment's use of static representations. The benchmarks are appealing to designers who find quantities to provide more useful guidance than the CDs' reliance solely on vocabulary items. The benchmarks are of three sorts: (1) binary (yes/no) measurements reflecting the presence (denoted S_p) of the elements of a static representation S , (2) measurements of the extent of characteristics (denoted S_c) in a language environment's static representation, or (3) number of user navigational actions (denoted NI) required to navigate to an element of the static representation if it is not already on the screen.

The benchmarks are given in Table 2. Note that there is no inherent "goodness" about a higher or a lower score in some of the benchmarks. Rather, it is up to the designer to decide whether a benchmark result of, say 3/4 for SN_1 , is better than 1/4 for SN_1 . Once having made that decision, designers can then quantitatively track their progress in the desired direction as they change their design decisions.

5. Conclusion

Programming and software engineering devices cannot be brought to end users by merely grafting colored graphics onto traditional approaches. The examples of end-user programming and software engineering systems presented here demonstrate the kinds of strategies that are instead employed in supporting end users to successfully problem solve about a program's logic. The strategies' underlying principles can be understood by turning to the relevant literature from the field of human-computer interaction.

The HCI work that has had the most influence so far has been the Cognitive Dimensions. Using Cognitive Dimensions (and/or their relative, Representation Design Benchmarks), it is possible to spot, very early in the design cycle, problems that will prevent end users from succeeding at using a proposed system. Spotting such problems early is critical, because early in the design cycle it is still possible to make fundamental changes. Waiting until late in a system's development cycle can doom the designer to being able to make only surface-level changes, which cannot correct fundamental flaws.

In summary, to be successful at such efforts, designers must recognize the fact that languages and software engineering tools intended for end users must be designed from the very beginning with human problem solving principles in mind.

Acknowledgments

We thank the Institute of Computer Based Software Methodology and Technology, Catena Corporation, and Iwate Prefectural University for their support of the project to which this paper relates.

References

- [1] L. Beckwith, M. Burnett, and C. Cook, Reasoning about Many-to-Many Requirement Relationships in Spreadsheets, *IEEE Symp. Human-Centric Computing Languages and Environments*, Arlington, VA, Sept. 2002 (to appear).
- [2] D. Brown, M. Burnett, and G. Rothermel, End-User Testing of Lyee Programs: A Preliminary Report, *International Workshop on Lyee Methodology*, Paris, France, Oct. 2002 (to appear).
- [3] M. Burnett, J. Atwood, R/ Djang, H. Gottfried, J. Reichwein, and S. Yang, Forms/3: A First-Order Visual Language to Explore the Boundaries of the Spreadsheet Paradigm, *Journal of Functional Programming* (Mar. 2001) pp. 155-206.
- [4] M. Burnett, S. Chekka, R. Pandey, FAR: An End-User Language to Support Cottage E-Services, *IEEE*

- Symposium on Human-Centric Languages*, Stresa, Italy, Sept. 2001, pp. 195-202.
- [5] T. Green and M. Petre, Usability Analysis of Visual Programming Environments: A 'Cognitive Dimensions' Framework, *Journal of Visual Languages and Computing* 7 (June 1996) pp. 131-174.
 - [6] N. Heger, A. Cypher, and D. Smith, Cocoa at the Visual Programming Challenge 1997, *Journal of Visual Languages and Computing* 9 (April 1998) pp. 151-169.
 - [7] E. Hutchins, J. Hollan, and D. Norman, Direct Manipulation Interfaces. In D. Norman and S. Draper (eds.), *User Centered System Design: New Perspectives on Human-Computer Interaction*. Lawrence Erlbaum Assoc., Hillsdale, NJ, 1986, pp. 87-124.
 - [8] D. Kurlander, Chimera: Example-Based Graphical Editing. In A. Cypher (ed.), *Watch What I Do: Programming by Demonstration*, MIT Press, Cambridge, MA, 1993.
 - [9] J. Landay and B. Myers, Sketching Interfaces: Toward More Human Interface Design, *Computer* 34 (March 2001) pp. 56-64.
 - [10] F. Modugno, A. Corbett, and B. Myers, Evaluating Program Representation in a Demonstrational Visual Shell, *Empirical Studies of Programmers: Sixth Workshop*, Alexandria, Virginia, January 1996, pp. 131-146.
 - [11] B. Nardi, *A Small Matter of Programming: Perspectives on End-User Computing*, MIT Press, Cambridge, MA, 1993.
 - [12] F. Negoro and I. Hamid, A Proposal for Intention Engineering, *International conference on Advances in Infrastructure for Electronic Business, Science, and Education on the Internet*, 2001.
 - [13] J. Reichwein, G. Rothermel, and M. Burnett, Slicing Spreadsheets: An Integrated Methodology for Spreadsheet Testing and Debugging. *Conference on Domain-Specific Languages*. Austin, Texas, October, 1999, pp. 25-38.
 - [14] G. Rothermel, M. Burnett, L. Li, C. DuPuis, and A. Sheretov, A Methodology for Testing Spreadsheets, *ACM Trans. Software Engineering and Methodology*, Jan. 2001, pp. 110-147.
 - [15] G. Rothermel, L. Li, C. DuPuis, and M. Burnett, What You See Is What You Test: A Methodology for Testing Form-Based Visual Programs, *Proc. Int'l. Conf. Software Engineering*, Apr. 1998, pp. 198-207.
 - [16] K. Rothermel, C. Cook, M. Burnett, J. Schonfeld, T. Green, and G. Rothermel, An Empirical Evaluation of a Methodology for Testing Spreadsheets, *Proc. Int'l. Conf. Software Engineering*, June 2000, pp. 230-239.
 - [17] C. Salinesi, M. Ben Ayed, and S. Nurcan, Development Using LYEE: A Case Study with LYEEALL, Technical Report TR1-2, The Institute of Computer Based Software Methodology and Technology, Oct. 2001.
 - [18] C. Salinesi, C. Souveyet, and R. Kla, Requirements Modeling in Lyee, Technical Report TR2-1, The Institute of Computer Based Software Methodology and Technology, Mar. 2001.
 - [19] B. Shneiderman, Direct Manipulation: A Step Beyond Programming Languages, *Computer* 16 (August 1983) pp. 57-69.
 - [20] D. Smith, A. Cypher, and J. Spohrer, Kidsim: Programming Agents without a Programming Language, *Communications of the ACM* 37 (July 1994) pp. 54-67.
 - [21] S. Tanimoto, VIVA: A Visual Language for Image Processing, *Journal of Visual Languages Computing* 2 (June 1990) pp. 127-139.
 - [22] C. Wallace, C. Cook, J. Summet, and M. Burnett, Assertions in End-User Software Engineering: A Think-Aloud Study, *IEEE Symp. Human-Centric Computing Languages and Environments*, Arlington, VA, Sept. 2002 (Tech. Note, to appear).
 - [23] S. Yang, M. Burnett, E. DeKoven, and M. Zloof, Representation Design Benchmarks: A Design-Time Aid for VPL Navigable Static Representations, *Journal of Visual Languages and Computing* 8 (October/December 1997) pp. 563-599.

Chapter 5

Human Factor and User Intention Capturing in Software

This page intentionally left blank

End-User Testing of Lyee Programs: A Preliminary Report

Darren BROWN, Margaret BURNETT, and Gregg ROTHERMEL
Department of Computer Science, Oregon State University, Corvallis, OR 97331 USA

Abstract. End-user specification of Lyee programs is one goal envisioned by the Lyee methodology. But with any software development effort comes the possibility of faults. Thus, providing end users a means to enter their own specifications is not enough; they must also be provided with the means to find faults in their specifications, in a manner that is appropriate not only for the end user's programming environment but also for his or her background. We have begun research into how to do exactly that. In this paper, we report our progress in investigating two possible testing approaches for end users who specify their own Lyee programs.

1. Introduction

One goal envisioned by the inventors of the Lyee methodology [25] is that even an end user will someday be able to enter a set of software requirements—without the assistance of a professional software developer—and that the methodology will be able to generate software that conforms to these requirements. (Following convention in the literature, the term *end users* as used here refers to people who are not professional programmers.) Indeed, other collaborators at this workshop have been working on important fundamentals that will contribute to this goal (e.g., [33, 34]). For this goal to become a reality, many questions relating to the human aspects of this goal must also be investigated, such as:

- Is it feasible to expect end users to enter software requirements at all?
- If so, to what level of detail must they descend to create requirements that are complete enough to generate the desired software?
- How can we ensure that requirements entered by such users are non-contradictory?
- Can end users understand the implications of the requirements they are entering well enough to recognize errors and correct them?

There is preliminary research contributing partial answers to the above questions, some of which we discuss in the companion paper [5]. These partial results allow cautious optimism that the questions above will be resolved. With this in mind, we will make the rather large assumption that these and related issues can be solved for the Lyee methodology. But once they have been solved, at least one question remains:

- How can end users test the requirements they enter? Specifically, can a testing methodology we have previously developed for end users, known as the WYSIWYT methodology, be adapted to the Lyee environment?

Addressing this question is our role in the Lyee collaboration project. In Section 2 we discuss relevant research, including a summary of the WYSIWYT methodology which we have previously devised for the spreadsheet paradigm. Section 3 presents a sketch of

attributes of a user requirements specification approach. Section 4 draws upon these attributes to consider formal models and test adequacy criteria upon which a testing methodology for Lye could rest. Section 5 summarizes the next steps in our investigation of end-user testing of Lye requirements.

We are latecomers to this collaboration, and our work began less than two months ago. This paper is therefore only a very early report of our progress.

2. Background

2.1 The WYSIWYT Testing Methodology

In previous work [29, 30, 31], we presented a testing methodology for spreadsheets termed the “What You See Is What You Test” (WYSIWYT) methodology. The WYSIWYT methodology provides feedback about the “testedness” of cells in spreadsheets in a manner that is incremental, responsive, and entirely visual. It is aimed at a wide range of spreadsheet users, including both end users and professional programmers. We have performed extensive empirical work, and our results consistently show that both end users and programmers test more effectively and efficiently using WYSIWYT than they do unaided by WYSIWYT (e.g., [11, 20, 29, 32]).

This proven effectiveness of WYSIWYT for spreadsheets suggests WYSIWYT as a possibility for testing in the Lye methodology, provided that it can be effectively adapted to Lye. In this section, we summarize the WYSIWYT methodology.

The underlying assumption behind the WYSIWYT methodology has been that, as the user develops a spreadsheet incrementally, he or she could also be testing incrementally. We have integrated a prototype of WYSIWYT into our research spreadsheet language Forms/3 [6, 7]. In our prototype, each cell in the spreadsheet is considered to be untested when it is first created, except *input cells* (cells whose formulas may contain constants and operators, but no cell references or *if*-expressions), which do not require testing. For the non-input cells, testedness is reflected via border colors on a continuum from untested (red) to tested (blue).

Figure 1 shows a spreadsheet used to calculate student grades in Forms/3. The spreadsheet lists several students, and several assignments performed by those students. The last row in the spreadsheet calculates average scores for each assignment, the rightmost column calculates weighted averages for each student, and the bottom right cell gives the overall course average (formulas not shown). With WYSIWYT, the process of testing spreadsheets such as the one in Figure 1 is as follows. During the user’s spreadsheet development, whenever the user notices a correct value, he or she lets the system know of this decision by *validating* the correct cell (clicking in the decision checkbox in its right corner), which causes a checkmark to appear, as shown in Figure 1. This communication lets the system track judgments of correctness, propagate the implications of these judgments to cells that contributed to the computation of the validated cell’s value, and reflect this increase in testedness by coloring borders of the checked cell and its contributing cells more tested (more blue). On the other hand, whenever the user notices an incorrect value, rather than checking it off, he or she eventually finds the faulty formula and fixes it. This formula edit means that affected cells will now have to be re-tested; the system is aware of which ones those are, and re-colors their borders more untested (more red). In this document, we depict red as light gray, blue as black, and the colors between the red and blue endpoints of the continuum as shades of gray.

WYSIWYT is based on an abstract testing model we developed for spreadsheets called a *cell relation graph* (CRG) [30]. A CRG is a pair (V, E) , where V is a set of formula graphs and E is a set of directed edges modeling dataflow relationships between pairs of

	NAME	ID	HWAVG	MIDTERM	FINAL	COURSE
1	Abbott, Mike	1035	89	91	86	89
2	Farnes, Joan	7649	92	94	96	94
3	Green, Matt	2314	78	80	75	78
4	Smith, Scott	2316	84	90	86	87
5	Thomas, Sue	9857	91	87	90	90
AVERAGE			87	88	87	88

Figure 1. Visual depiction of testedness of a student grades spreadsheet. Blue-bordered cells (black in this paper) such as the first cell in the *Average* row are tested, red-bordered cells (light gray in this paper) such as the second cell in the *Average* row are untested, and shades between such as the top cell in the *Course* column are partially tested. The upper right corner of each spreadsheet reports a spreadsheet's overall testedness percentage. Checkmarks were placed by the user to indicate that a value is correct, and question marks point out the cells in which checkmarks would increase testedness according to the adequacy criterion.

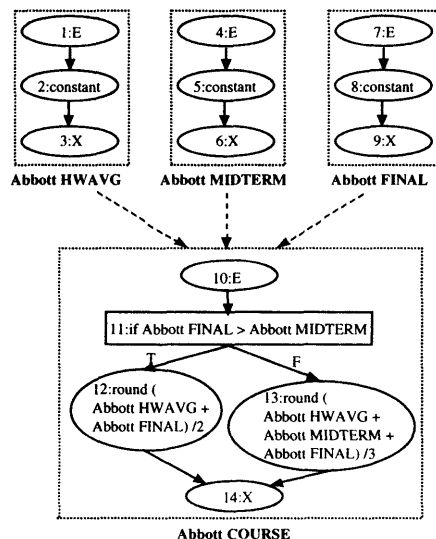


Figure 2. A partial cell relation graph of Figure 1. These are the formula graphs for the top row ("Abbott, Mike"). Dashed arrows indicate dataflow edges between cells' formula nodes. For clarity in this figure, we preface cell names with "Abbott" instead of with the internal IDs actually used.

elements in V. A formula graph models flow of control within a single cell's formula, and is comparable to a control flow graph. In simple spreadsheets, there is one formula graph for each cell. (See [8, 9] for discussions of how complex spreadsheets are treated.) For example, Figure 2 shows a portion of the CRG for the cells in Figure 1, delimited by dotted rectangles. The process of translating an abstract syntax tree representation of an expression into its control flow graph representation is well known [1]; a similar translation applied to the abstract syntax tree for each formula in a spreadsheet yields that formula's formula graph. In these graphs, nodes labeled "E" and "X" are *entry* and *exit* nodes, respectively, and represent initiation and termination of evaluation of formulas. Nodes with multiple out-edges are *predicate* nodes (represented as rectangles). Other nodes are *computation* nodes. Edges within formula graphs represent flow of control between expressions, and edge labels indicate the value to which condition expressions must evaluate for particular branches to be taken.

We used the cell relation graph model to define several test adequacy criteria for spreadsheets [30]. (A *test adequacy criterion* [36] is a definition of what it means for a program to be tested "enough.") The strongest criterion we defined, *du-adequacy*, is the criterion we have chosen for our work. The *du-adequacy criterion* is a type of dataflow adequacy criterion [15, 23, 28, 37]. Such criteria relate test adequacy to interactions between definitions and uses of variables in source code (*definition-use associations*, abbreviated *du-associations*). In spreadsheets, cells play the role of variables; a *definition* of cell *c* is a node in the formula graph for *c* representing an expression that defines *c*'s value, and a *use* of cell *c* is either a computational use (a non-predicate node that refers to *c*) or a predicate use (an out-edge from a predicate node that refers to *c*). For example, in Figure 2, nodes 2, 5, 8, 12, and 13 are definitions of their respective cells, nodes 12 and 13 are computational uses of the cells referenced in their expressions, and edges (11,12) and (11,13) are predicate uses of the cells referenced in predicate node 11. Under this criterion, a cell *x* will be said to have been tested enough when all of its definition-use associations have been *covered* (executed) by at least one test. In this model, a *test* is a user decision as to whether a particular cell contains the correct value, given the inputs upon which it depends. Decisions are communicated to the system when the user checks off a cell to validate it. Thus, given a cell *x* that references *y*, *du-adequacy* is achieved with respect to the interactions between *x* and *y* when each of *x*'s *uses* (references to *y*) of each *definition* in *y* has been covered by a test.

Thus, if the user manages to turn all the red (light gray) borders blue (black), the *du-adequacy* criterion has been satisfied. This may not be achievable, since not all *du-associations* are executable in some spreadsheets (termed *infeasible*). Even so, in our empirical work, subjects have been significantly more likely to achieve *du-adequate* coverage and do so efficiently using the WYSIWYT methodology than those not using it [20, 32], *du-adequate* test suites have frequently been significantly more effective at fault detection than random test suites [29], and subjects have been significantly more likely to correctly eliminate faults using the WYSIWYT methodology than those not using it [11]. Both programmer [11, 32] and end-user [20] audiences have been studied.

2.2 Testing Approaches Based on Finite State Machines

The WYSIWYT methodology is based on an underlying dataflow test adequacy criterion, but other criteria could also be employed, even while retaining WYSIWYT's emphasis on incremental testing, visual devices, and immediate feedback. One family of criteria of interest involves testing of finite state machines.

A finite state machine (FSM) contains a finite number of states and produces outputs on state transitions after receiving inputs [24]. FSMs have been used to model system behavior in various ways. There are applications of FSMs in which the model is intended to represent exactly the modeled system. For example, in protocol specification testing, networking protocol models are represented with FSMs and represent the complete protocol. However, most FSMs used to represent systems are an abstraction of the intended system to some degree. In such cases, even if it were possible to fully test an FSM prior to the creation of the program, this will not guarantee adequate testing of the system after it has been created.

Previous work on testing FSMs tries to account for information missed during the modeling process [24]. Also researched heavily is the amount of abstraction that a model should exercise [38]. This is obviously not an issue for testing a precise model that completely specifies the program, but there is still much that can be learned about the testing approaches and techniques associated with imprecise models. One particularly relevant point is that FSMs are most often used only to test the control structure of a program, complementing other testing approaches used in testing other aspects of the program [10].

This suggests the possibility that dataflow-based and state-based test adequacy criteria might be combined to complement one another, such that each provides unique testedness information. Such an approach could draw from both WYSIWYT and from previous research on state-based testing approaches, such as [4, 13, 16, 17, 24, 35]. Part of our research plan is to investigate this strategy.

3. Attributes of End-User Requirements Specification

To define an end-user methodology for testing Lyee specifications, we first need to establish basic attributes of what the end users will be testing.

The current emphasis by the developers of Lyee on *screen transition diagrams* provided by end users suggests using that device for actual entry by end users of their requirement specifications. There is related research supporting the viability of this approach: Landay and Myers have devised an approach based on such diagrams to allow users to create graphical user interfaces [21, 22]. The general idea is that a user can design an interface by explicitly sketching how the intended interface is to be used. See Figure 3. In this paper, we will assume that this idea can be applied to end-user programming for the Lyee methodology.

We are also assuming that end users will enter a *complete* set of requirements, without the help of a professional developer. Thus, testing the screen transition diagram is testing the program—because there will be no information in the program that was not generated by the user's screen transition diagrams. (We simply assume that this is true for now; a later collaboration may design exactly how this will work.)

From the assumption that end users will provide complete specifications via screen transition diagrams, how can testing be supported? The first issue to consider is how testing support will be presented to users. Our approach will address this issue in a way that satisfies two constraints: (1) the presentation of testedness will be integrated with the screen transition diagrams, as in the WYSIWYT methodology, and (2) any update or test made by the user will be immediately and visually reflected in the presentation.

The second issue to consider is, what is the user going to test? In order to consider this issue, it is necessary to define further assumptions about the user's requirement specification elements.

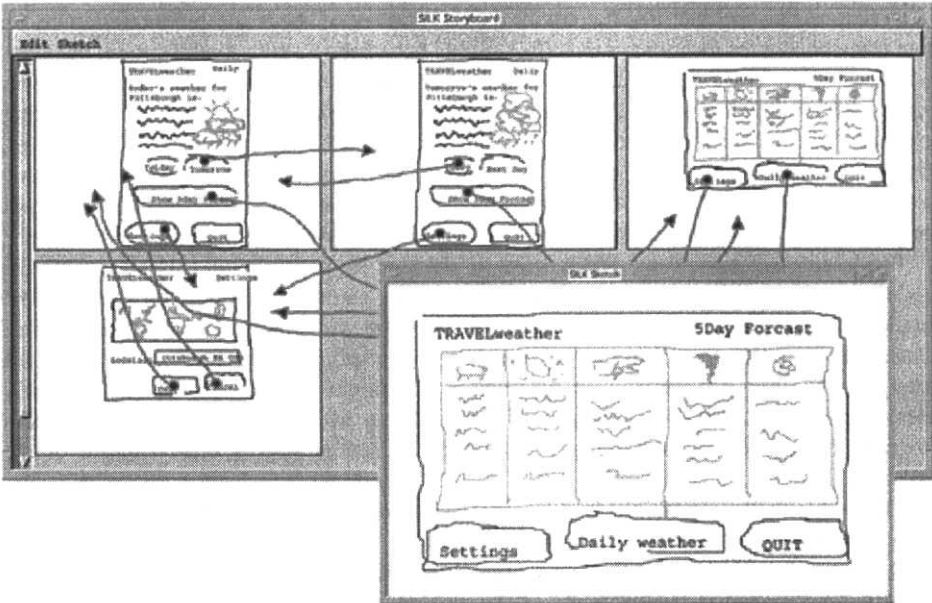


Figure 3: A SILK sketch (front) of a five-day weather forecast and storyboard (rear) [22]. An experienced user-interface designer created the sketch. The designer has also created buttons and drawn arrows to show the screen transitions that occur when the user presses the buttons.

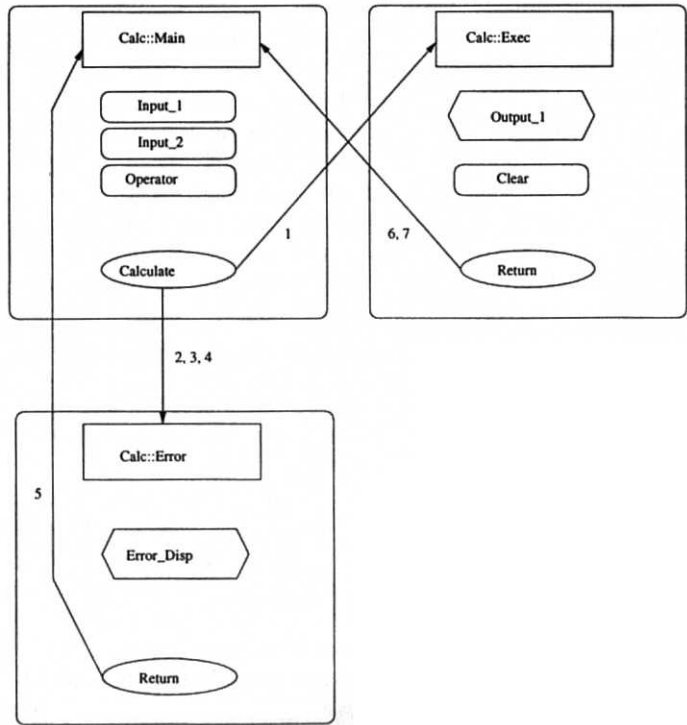


Figure 4: An example screen transition diagram.

With screen transition diagrams, the user communicates a program using screens, objects on those screens, and transition-actions. (It should be understood that we are attempting to relate only *what* the user communicates, not *how*.) Thus, our next assumption is that a screen is a collection of objects, which will be used and/or produced by computations. In Figure 4, a screen is denoted as a large rounded rectangle containing the screen name and the objects that are on the screen. The general idea of how this technology can come together with the Lyee methodology is illustrated in Figure 5.

The diagram of Figure 4 depicts requirements for a program named *calc*, which takes two numbers and an operator, and then returns the operator applied to the two numbers as long as the numbers and the operator are valid. (At the moment, this is just a hand-drawn depiction of the idea; in practice the style of the diagram is expected to be similar to the style of Figure 3.) In the diagram, input objects are denoted by soft-cornered rectangles within the screens. For example, in the screen *Calc::Main*, there are the three input objects: *Input_1*, *Input_2*, and *Operator*. An event object (an object capable of generating events) is shown with an oval as with *Calculate*. Transitions to screens are shown by connecting an event object to the title of the screen that the event yields with a directed arrow, such as

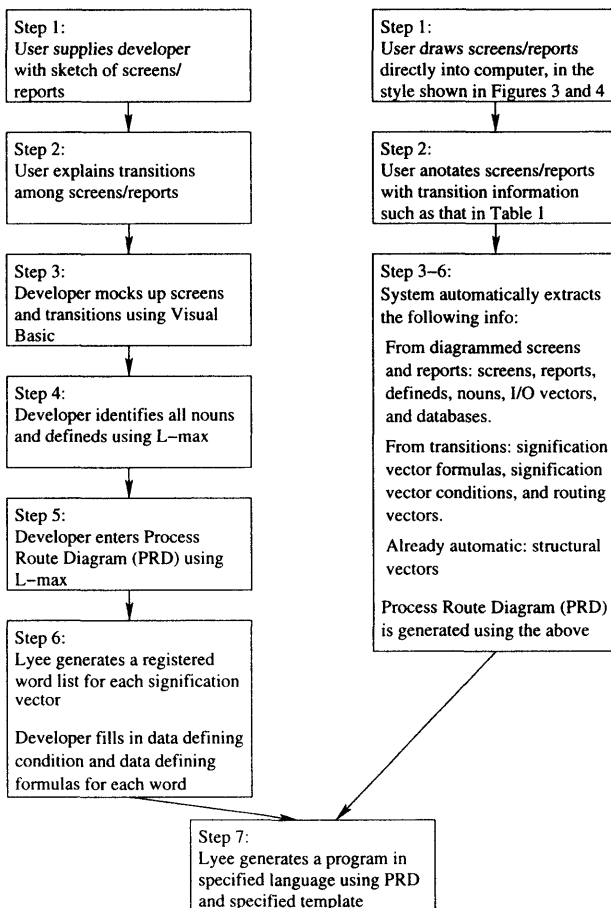


Figure 5: A strategy (right side) for applying end-user programming to the current (left side) Lyee methodology (also assumes incorporation of work by other collaborators in relevant ways).

Table 1: Transitions/actions. ("NAN" stands for "not a number").

Transition number	Screen	Event	Condition(s)	Destination Screen	Action(s)
1	Calc::Main	Pressed <i>calculate</i> object	AND ((input_1 != NAN) (input_2 != NAN) (OR (operator != /) (AND (operator == /) (input_2 != 0))))	Calc::Exec	output_1 = input_1 operator input_2
2	Calc::Main	Pressed <i>calculate</i> object	(input_1 == NAN)	Calc::Error	Error_Disp = "input 1 is not a number"
3	Calc::Main	Pressed <i>calculate</i> object	(input_2 == NAN)	Calc::Error	Error_Disp = "input 2 is not a number"
4	Calc::Main	Pressed <i>calculate</i> object	AND ((operator == /) (input_2 == 0))	Calc::Error	Error_Disp = "divide by zero"
5	Calc::Error	Pressed <i>return</i> object		Calc::Main	
6	Calc::Exec	Pressed <i>return</i> object	(clear == "yes")	Calc::Main	input_1 = "null" input_2 = "null" operator = "null"
7	Calc::Exec	Pressed <i>return</i> object	OR ((clear == "no") (clear == "null"))	Calc::Main	

(*Calculate*, *Calc::Exec*). Finally, output objects are denoted with a hexagon shape, as with *Output_1*. Transitions are defined in Table 1 and referred to by the numbers in the table.

The assumptions about action objects are as follows. Action objects are what describe and cause computations to occur. They consist of an event, a set of conditions, a destination screen, and a set of actions to be taken. Actions include Lyee's notion of formulas, and can reference both input and output objects; they can also affect both input and output objects.

We emphasize that the format shown in Table 1 is only for precision of this discussion, and is not suitable for end users. As Pane and Myers showed empirically [26], end users are not very successful at using Boolean AND and OR, and do not tend to understand the use of parentheses as ways to specify precedence. They suggest some alternatives to these constructs, and empirically show that end users can use one set of such alternatives successfully [27]. In addition to their suggested alternatives, other possibilities include the demonstrational rewrite rules of Cocoa [18] or Visual AgentTalk [19], which have been demonstrated to be usable by end users.

As the above example illustrates in part, our next assumption is that there are three types of objects: input, output, and action. A specialized kind of input object is an event object.

which generates a user event if the user interacts with it. Our assumption about input objects is that they allow the user to enter input, via the keyboard or the mouse, but they are also updatable by the program. An assumption about output objects is that they cannot receive user inputs. Instead, their purpose is to receive the results of computations, but they can also provide input values to computations.

Action objects are a slightly more powerful form of transition than is traditional, but in this document we use the terms interchangeably. The difference is that the event and conditions that are required to execute the actions are only a *partial* specification of state; however, the destination screen and the actions to execute upon taking the transition completely specify a new state. A transition from and to the same screen is legal.

Once an event occurs, the actions are performed and the destination screen is displayed. Those actions performed are the ones in the action object that matches the satisfied conditions. Since this is a deterministic machine, only one transition can be allowed to be possible given an event and conditions. Some possible ways to accomplish this are to require that conditions for a given event are mutually exclusive, that transitions must be prioritized, or that the first transition that meets the criterion for transition will be taken without regard for further applicable transitions.

Events need not require user intervention. For example, an event may be defined as a certain output cell reaching some threshold, or simply the current screen being reached on a given transition.

To summarize this section, we have identified a set of attributes that we assume to be present in a future approach to allowing end users to enter their requirements into the Lyee set of tools. Resting upon these assumptions allows consideration into how end users might test such requirements.

4. Testing End-User Requirements Specifications

Given the above assumptions about end users' requirement specifications via screen transition mechanisms, we are developing two approaches to testing. One approach is based on the idea of WYSIWYT and the other is based on approaches for state-based testing. We are investigating the alternative approaches of end-user testing of Lyee requirements specifications using each of these approaches singly and in combination.

4.1 WYSIWYT-Based Testing

To make use of the WYSIWYT methodology for testing end-user spreadsheets, we can model a screen transition diagram as a set of du-associations associated with each transition. We believe (but have not yet formally proven) that this model will support testing of all of the cases possible under the assumptions stated in this paper, except for transitions into screens that contain no uses.

The following define the definitions and uses in this model:

A definition of input object A is:

- the specification of A as an input value (including its initial value and any future values input), or
- an assignment to A in an action (presence of A in the action's left-hand side).

A definition of output object A is:

- the specification of A's initial value, or
- an assignment to A in an action (presence of A in the action's left-hand side).

A use of object A is:

- a reference to A the right-hand side of an action, or
- a reference to A in a condition.

Building upon these definitions in the same manner as in Section 2, *du-associations* are interactions between definitions and uses, and the other definitions follow. Using this model, WYSIWYT's dataflow-based testing techniques can be employed to try to cover each du-association. The system will treat the user as an oracle and allow the user to state whether, given a particular scenario of inputs, results are correct. Figure 6 sketches the algorithm for this aspect of the methodology. For now we will adopt WYSIWYT's notion that testing is adequate when all feasible definition-use associations have been exercised by at least one test.

The algorithm depicts only one testing situation, namely that in which the user has asked the system to suggest possible input values (termed the "Help Me Test" feature in our Forms/3 prototype of WYSIWYT) [14]. As in our previous work, this feature will work in tandem with the user's ability to specify their own input values when they prefer. Also as in the original WYSIWYT methodology, feedback given to the user under this approach will be entirely visual, using devices such as those used in WYSIWYT. We are currently in the process of analyzing empirical data about the way end users make use of "Help Me Test"; early indications suggest that the feature positively impacts users' abilities to find faults.

Pick a transition T; let O be the set of objects with uses in T.
 For each object o in O, find each definition d for which du-associations (d,u) are not yet covered. Set new specific values for each such definition d.
 Keep setting definitions' values until T is traversed.
 Allow Oracle to say if, given current definitions and transition T, the output is correct.

Figure 6: Algorithm sketch for one run of a WYSIWYT test generator for screen transition diagrams.

Transition 2:
 Uncovered DU's: 7, 8; DU Picked: 7.
 1st try: input1 = 3, input2 = NAN, operator = *, can't cover.
 2nd try: input1 = NAN, input2 = NAN, operator = *, T covered.
 Oracle validates.
 DU's covered so far: 7.
 Infeasible DU's tried to cover: none.

Transition 2:
 Uncovered DU's: 8; DU Picked: 8.
 1st try: input1 = null, input2 = null, operator = null, T covered.
 Oracle validates.
 DU's covered so far: 7, 8.
 Infeasible DU's tried to cover: none.

Transition 1:
 Uncovered DU's: 1, 2, 3, 4, 5, 6, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24; DU Picked: 15.
 1st try: input1 = NAN, input2 = 4, operator = +, can't cover.
 2nd try: input1 = 3, input2 = NAN, operator = *, can't cover.
 3rd try: input1 = 3, input2 = NAN, operator = /, can't cover.
 4th try: input1 = 3, input2 = 4, operator = +, T covered.
 Oracle validates.
 DU's covered so far: 1, 3, 5, 7, 8, 15, 17, 19, 21, 23.
 Infeasible DU's tried to cover: none.

Transition 1:
 Uncovered DU's: 2, 4, 6, 16, 18, 20, 22, 24; DU Picked: 16.
 1st try: input1 = null, input2 = null, operator = null, can't cover. (cannot manipulate constants).
 DU's covered so far: 1, 3, 5, 7, 8, 15, 17, 19, 21, 23.
 Infeasible DU's tried to cover: 2, 4, 6, 16, 18, 20, 22, 24.

... (and so on)...

Figure 7: The beginning of one possible run of a WYSIWYT-based test generation.

Table 2: Definition-use associations in the calc example.

DU-association number	Definition	Use
1	Input_1 (as program input)	Transition 1 action
2	Input_1 (resulting from Transition 6 action)	Transition 1 action
3	Input_2 (as program input)	Transition 1 action
4	Input_2 (resulting from Transition 6 action)	Transition 1 action
5	Operator (as program input)	Transition 1 action
6	Operator (resulting from Transition 6 action)	Transition 1 action
7	Input_1 (as program input)	Transition 2 condition
8	Input_1 (resulting from Transition 6 action)	Transition 2 condition
9	Input_2 (as program input)	Transition 3 condition
10	Input_2 (resulting from Transition 6 action)	Transition 3 condition
11	Operator (as program input)	Transition 4 condition
12	Operator (resulting from Transition 6 action)	Transition 4 condition
13	Input_2 (as program input)	Transition 4 condition
14	Input_2 (resulting from Transition 6 action)	Transition 4 condition
15	Input_1 (as program input)	Transition 1 condition
16	Input_1 (resulting from Transition 6 action)	Transition 1 condition
17	Input_2 (as program input)	Transition 1 condition (1st occurrence)
18	Input_2 (resulting from Transition 6 action)	Transition 1 condition (1st occurrence)
19	Operator (as program input)	Transition 1 condition (1st occurrence)
20	Operator (resulting from Transition 6 action)	Transition 1 condition (1st occurrence)
21	Operator (as program input)	Transition 1 condition (2nd occurrence)
22	Operator (resulting from Transition 6 action)	Transition 1 condition (2nd occurrence)
23	Input_2 (as program input)	Transition 1 condition (2nd occurrence)
24	Input_2 (resulting from Transition 6 action)	Transition 1 condition (2nd occurrence)
25	Clear (as program input)	Transition 6 condition (1st occurrence)
26	Clear (as program input)	Transition 6 condition (2nd occurrence)
27	Clear (as program input)	Transition 7 condition

Note that the algorithm *allows* the user to validate the value, but it does not actually *require* the user to do anything. This is an important aspect of our approach, and follows principles implied in Blackwell's end-user programming model of attention investment [3]. That is, our approach does not attempt to alter the user's work priorities by requiring them to answer dialogues about correctness, because the user might find that counter-productive and stop using the feature. Rather, our approach provides *opportunities* (through decorating the diagrams with clickable objects) for the user to provide information if they choose. This allows the user to take the initiative to validate, but does not interrupt them from their current processes by requiring information at any particular time. For example, this allows the user to fix a fault as soon as a test reveals it, if they immediately spot the cause, rather than requiring them to continue with additional test values.

For example, in Figure 4, there are 27 du-associations, which are listed in Table 2. The system begins by picking a transition, and tries to cover the du-associations associated with that transition. To do so, it sets values of associated definitions until the condition for taking the transition is met. It then visibly executes the program given these values, and provides the user an opportunity to pronounce the demonstrated behavior correct for these inputs (i.e., to validate). If the user validates, then the objects, du-associations, and transitions are colored closer to the testedness color (blue in our previous prototypes). One possible run of this method on *calc* might start out as in the sequence of Figure 7.

Pick a transition T from screen S_i on a set of input objects A to screen S_j of machine M .
 Set a sequence of input values that transfer the machine M from its current screen to S_i .
 Set A 's objects to input values that satisfy T 's conditions.
 Allow Oracle to say if, given the values in A , the behavior is correct.

Figure 8: Algorithm sketch for one run of a transition-adequate test generator for screen transition diagrams.

Current Screen: Calc::Main Transition 6: Path: Transition 1 (Calculate, <Input_1 = 3, Input_2 = 4, Operator = +>), Transition 6 (Return, <Clear = "no">). Oracle validates. Transitions covered so far: 1, 6.
Current Screen: Calc::Main Transition 4: Path: Transition 4 (Calculate, <Input_1 = 4, Input_2 = 0, Operator = />). Oracle validates. Transitions covered so far: 1, 4, 6.
Current Screen: Calc::Error Transition 3: Path: Transition 5 (Return, <>), Transition 3 (Calculate, <Input_1 = 3, Input_2 = NAN, Operator = *>). Oracle validates. Transitions covered so far: 1, 3, 4, 5, 6.
Current Screen: Calc::Error Transition 6: Path: Transition 5 (Return, <>), Transition 1 (Calculate, <Input_1 = 7, Input_2 = 13, Operator = ->), Transition 6 (Return, <Clear = "null">). Oracle validates. Transitions covered so far: 1, 3, 4, 5, 6.
Current Screen: Calc::Main Transition 1: Path: Transition 1 (Calculate, <Input_1 = 6, Input_2 = 1, Operator = +>). Oracle validates. Transitions covered so far: 1, 3, 4, 5, 6.
Current Screen: Calc::Exec Transition 2: Path: Transition 7 (Return, <Clear = "yes">), Transition 2 (Calculate, <Input_1 = NAN, Input_2 = 7, Operator = />). Oracle validates. Transitions covered so far: 1, 2, 3, 4, 5, 6, 7.

Figure 9: One possible run of a state-based test generation algorithm on the *calc* example. An executed transition is denoted (action object pressed, <conditions>).

4.2 State-Based Testing

An alternative to WYSIWYT's dataflow-based approach is to test in such a way that various combinations of transitions in a screen transition diagram are exercised. In this context we model a screen transition diagram with a modified state transition diagram. A transition will bring the program to a new screen and alter information within this new screen. One possible adequacy criterion might be that each possible transition between screens must be covered by a test (*transition adequacy*). We will begin our discussion using this adequacy criterion. Another possible adequacy criterion would be that some

combination of “tours” through the diagram is required; we briefly discuss this alternative later. As with WYSIWYT, our approach to state-based testing is interactive and visual, and relies on the user as an oracle. Given these assumptions, it is also necessary, as it was with WYSIWYT, to define a *test* as a user decision of correctness that has been explicitly communicated to the system (termed a *validation*).

Figure 8 sketches a test generation algorithm under transition adequacy. One possible run of this method on *calc* might execute in the sequence of Figure 9. After picking a transition, the system selects a path and inputs that allow it to reach the transition’s starting state from its current state, so as to make the transition. The approach uses some strategy for choosing transitions that allows computational efficiency and attempts to avoid disorienting the user. (The strategy is still being considered, but is likely to be a nearest-neighbor approach.) The system then makes the transition and allows the oracle to validate. Once all transitions have participated in a test (validation), the adequacy criterion is satisfied.

5. The Next Steps

Given the above two approaches to testing requirements specifications expressed as screen transition diagrams, we plan to explore the merits of both approaches, both singly and in combination. Both approaches are likely to have complementary strengths in helping to find faults. Thus, combining the dataflow-based criterion of WYSIWYT with the control flow elements of state-based testing may detect the most faults. However, there are a number of technical issues involved in this strategy. For example, should emphasis be placed on one of the methods over the other in defining the adequacy criterion?

If we ultimately choose a combination of the approaches, or even if we instead choose to simply support state-based testing, one of the important issues we must investigate is suitable adequacy criteria for the state-based testing. The simple example used transition adequacy. However, stronger criteria can include some form of tour coverage. A tour is a set of transitions required to move between two given states. Obviously, if there are cycles in the screen transition diagram, there are a potentially infinite number of tours between any two states. But there are methods for addressing this problem, such as covering only acyclic tours or restricting the number of times that the same cycle may be traversed. Tours force traversal of sequences of transitions, but there are likely to be too many arbitrary sequences to consider. It may be useful to switch between tour adequacy and transition adequacy, requiring tour adequacy for just those tours that are implied by dataflow pairs, and then switching to transition adequacy for the rest.

A critical factor in choosing which of these possible approaches to use (WYSIWYT, state-based, or a combination) and what adequacy criteria to employ, is whether we can devise a way to effectively communicate with the user about our testing methodology’s reasoning. From literature on on-line trust and its impact on the usefulness of on-line systems, it is clear that users need to understand the system’s logic to trust it, and that if they do not trust it, they will not bother to provide the system with the information (validations) needed by the system to help users test [2, 12]. The most important principle underlying this research is that the purpose of a testing methodology is to support end users’ ability to problem solve about the correctness of their requirement specifications; any design choices that we make must above all uphold this principle, even if it means sacrificing other desirable attributes. After all, if end users do not find it to be helpful, they will not use it or gain from it.

Acknowledgments

We thank the Institute of Computer Based Software Methodology and Technology, Catena Corporation, and Iwate Prefectural University for their support of this project.

References

- [1] A. Aho, R. Sethi, and J. Ullman, *Compilers*, Principles, Techniques, and Tools. ISBN: 0201100886. Addison-Wesley Publishing Company, Reading, MA, 1986.
- [2] N. Belkin, Helping People Find What They Don't Know, *Comm. ACM* **41** (Aug. 2000) pp. 58-61.
- [3] A. Blackwell and T. R. G. Green, Investment of Attention as an Analytic Approach to Cognitive Dimensions. In T. Green, R. Abdullah & P. Brna (Eds.) *Collected Papers Wkshp. Psychology of Programming Interest Group*, 1999, pp. 24-35.
- [4] E. Börger, A. Cavarra, and E. Riccobene, Modelling the Dynamics of UML State Machines, *International Workshop on Abstract State Machines*, 2000.
- [5] M. Burnett, Bringing HCI Research to Bear Upon End-User Requirement Specification, submitted to *International Workshop on the Lyee Methodology*, June 2002.
- [6] M. Burnett, J. Atwood, R. Djang, H. Gottfried, J. Reichwein, and S. Yang, Forms/3: A First-Order Visual Language to Explore the Boundaries of the Spreadsheet Paradigm, *Journal of Functional Programming* **11** (2001), pp. 155-206.
- [7] M. Burnett and H. Gottfried, Graphical Definitions: Expanding Spreadsheet Languages Through Direct Manipulation and Gestures, *ACM Trans. Computer-Human Interaction* **5** (1998), pp. 1-33.
- [8] M. Burnett, B. Ren, A. Ko, C. Cook, and G. Rothermel, Visually Testing Recursive Programs in Spreadsheet Languages, *IEEE Symp. Human-Centric Computing Languages and Environments*, Stresa, Italy, Sept. 5-7, 2001, pp. 288-295.
- [9] M. Burnett, A. Sheretov, B. Ren, and G. Rothermel, Testing Homogeneous Spreadsheet Grids with the 'What You See Is What You Test' Methodology, *IEEE Trans. Software Engineering* **28** (June 2002).
- [10] T. S. Chow, Testing Software Design Modeled by Finite-State Machines, *IEEE Trans. Software Engineering* **4** (May 1978), pp. 178-187.
- [11] C. Cook, K. Rothermel, M. Burnett, T. Adams, G. Rothermel, A. Sheretov, F. Cort, and J. Reichwein, Does a Visual 'Testedness' Methodology Aid Debugging? Oregon State University TR #99-60-07, rev. Mar. 2001. Available at [ftp://ftp.cs.orst.edu/pub/burnett/TR.EmpiricalTestingDebug.ps](http://ftp.cs.orst.edu/pub/burnett/TR.EmpiricalTestingDebug.ps)
- [12] C. Corritore, B. Kracher, and S. Wiedenbeck, Trust in the Online Environment, *HCI International, Vol. 1*, New Orleans, LA, Aug. 5-10 2001, pp. 1548-1552.
- [13] S. R. Dalal, A. Jain, N. Karunanithi, J. M. Leaton, C. M. Lott, G. C. Patton, Model-Based Testing in Practice, *Int'l. Conf. Software Engineering*, Apr. 1999, pp. 285-294.
- [14] M. Fisher II, M. Cao, G. Rothermel, C. Cook, and M. Burnett, Automated Test Generation for Spreadsheets, *Int'l. Conf. Software Engineering*, Orlando, FL, May 19-25, 2002, pp. 141-151.
- [15] P. Frankl and E. Weyuker, An Applicable Family of Data Flow Criteria, *IEEE Trans. Software Engineering* **14** (1988), pp. 1483-1498.
- [16] S. Fujiwara, G. v. Bochmann, F. Khendek, M. Amalou, and A. Ghedamsi, Test Selection Based on Finite State Models, *IEEE Trans. Software Engineering* **25** (June 1991), pp. 591-603.
- [17] A. Gargantini and C. Heitmeyer, Using Model Checking to Generate Tests from Requirements Specifications, *Proc. 7th European Engineering Conference and the 7th (ACM) (SIGSOFT) Symp. the Foundations of Software Engineering*, Sept. 1999, pp. 146-162.
- [18] N. Heger, A. Cypher, and D. Smith, Cocoa at the Visual Programming Challenge 1997, *Journal of Visual Languages and Computing* **9** (April 1998), 151-169.
- [19] A. Ioannidou and A. Repenning, End-User Programmable Simulations, *Dr. Dobb's Journal* **24** (August 1999), pp. 40-48.
- [20] V. Krishna, C. Cook, D. Keller, J. Cantrell, C. Wallace, M. Burnett, and G. Rothermel, Incorporating Incremental Validation and Impact Analysis into Spreadsheet Maintenance: An Empirical Study, *Int'l. Conf. Software Maintenance*, Florence, Italy, Nov. 2001, pp. 72-81.
- [21] J. Landay, Interactive Sketching for the Early Stages of User Interface Design, Ph.D. Dissertation, Tech. Report #CMU-CS-96-201, Computer Science Department, Carnegie Mellon University, Pittsburgh, PA, December, 1996.
- [22] J. Landay and B. Myers, Sketching Interfaces: Toward More Human Interface Design, *Computer* **34** (March 2001), 56-64.
- [23] J. Laski and B. Korel, A Data Flow Oriented Program Testing Strategy, *IEEE Trans. Software Engineering* **9** (May 1993), pp. 347-354.
- [24] D. Lee and M. Yannakakis, Principles and Methods of Testing Finite State Machines—A Survey, *Proc. IEEE*, August 1996, pp. 1090-1123.
- [25] F. Negro and I. Hamid, A Proposal for Intention Engineering, *Int'l. Conf. Advances in Infrastructure for Electronic Business, Science, and Education on the Internet*, 2001.
- [26] J. Pane and B. Myers, Tabular and Textual Methods for Selecting Objects From a Group, *IEEE Symp. Visual Languages*, Seattle, Washington, Sept. 10-13, 2000, pp. 157-164.
- [27] J. Pane, B. Myers, and L. Miller, Using HCI Techniques to Design a More Usable Programming System, *IEEE Human-Centric Computing Languages and Environments*, Arlington, VA, Sept. 2002 (to appear).

- [28] S. Rapps and E. Weyuker, Selecting Software Test Data Using Data Flow Information, *IEEE Trans. Software Engineering* **11** (1985), pp. 367-375.
- [29] G. Rothermel, M. Burnett, L. Li, C. DuPuis, and A. Sheretov, A Methodology for Testing Spreadsheets, *ACM Trans. Software Engineering and Methodology* **10** (Jan. 2001), pp. 110-147.
- [30] G. Rothermel, L. Li, and M. Burnett, Testing Strategies for Form-based Visual Programs, *International Symp. Software Reliability Engineering*, 1997, pp. 96-107.
- [31] G. Rothermel, L. Li, C. DuPuis, and M. Burnett, What You See Is What You Test: A Methodology for Testing Form-Based Visual Programs," *Int'l. Conf. Software Engineering*, Apr. 1998, pp. 198-207.
- [32] K. Rothermel, C. Cook, M. Burnett, J. Schonfeld, T. Green, G. Rothermel, An Empirical Evaluation of a Methodology for Testing Spreadsheets, *Proc. Int'l. Conf. Software Engineering*, June 2000, pp. 230-239.
- [33] C. Salinesi, M. Ben Ayed, and S. Nurcan, Development Using LYEE: A Case Study with LYEEALL, Technical Report TR1-2, Institute of Computer Based Software Methodology and Technology, Oct. 2001.
- [34] C. Salinesi, C. Souveyet, and R. Kla, Requirements Modeling in Lyee, Technical Report TR2-1, Institute of Computer Based Software Methodology and Technology, Mar. 2001.
- [35] Q. Tan, A. Petrenko and G. v. Bochmann, A Test Generation Tool for Specifications in the Form of State Machines, *Proc. International Communications Conference (ICC)* 96, Jun. 1996, pp.225-229.
- [36] E. Weyuker, Axiomatizing, Software Test Data Adequacy, *IEEE Trans. Software Engineering* **12** (1986), pp. 1128-1138.
- [37] E. Weyuker, More Experience with Dataflow Testing, *IEEE Trans. Software Engineering* **19** (Sept. 1993), pp. 912-919.
- [38] M. K. Zimmerman, K. Lundqvist, and N. Leveson, Investigating the Readability of State-Based Format Requirements Specification Languages, *24th Int'l. Conf. Software Engineering*, Orlando, FL, May 19-25, 2002, pp. 33-43.

Developing Multimedia Systems: the understanding of intention

Anthony Burgess

Department of Computer Science, Loughborough University, Leicestershire UK

Abstract - Multimedia CD-ROM production is in many ways different to conventional software development. This report explores the differences between the two development processes, and the issues that are involved. The Lyee software methodology will be explored, and its ideas will be applied to the development of multimedia systems, to examine its compatibility. The main issues that arise from this will be examined further to highlight the differences between the two development processes. Where appropriate, possible modifications to the Lyee software methodology will also be examined. Within multimedia CD-ROM production the role of the systems engineer will be examined to try and determine their role within the development process. The user within the system will also be examined, to examine the nature of their intentions, and how they may change during the development process.

1. Introduction

The development and design of multimedia systems is a difficult process. This is due to the nature of the material contained in these systems and because it is necessary to introduce a multimedia designer into the process to aid the user. There are many texts on how to manage a project, yet there are few that focus on the design process itself [1]. This report aims to find ways of helping the user to achieve their goals more easily with reduced help from the multimedia designer or Systems Engineer. It will explore the possibility of introducing a model to represent the development process, bearing in mind that trying to form a framework in which to model such systems could result in a model that is either too specific, that it can not be used on other systems, or too abstract that it fails to meet the need for which it was designed [2].

The Lyee Methodology was invented by Fumio Negoro, founder of the Institute of Computer Based Software Methodology and Technology. This methodology is patented in Japan, USA, Canada, Korea, New Zealand, Singapore, and Europe [3].

The Lyee methodology is different to conventional software methodologies. Using this methodology, many of the stages within the software development process can be removed [4]. Within Lyee, software specifications are not required. Here the aim is to generate an automatic agreement between user requirements and the programs that are produced, without the need for specifications to be written. This is achieved by choosing nouns from the requirements in natural language and making the same structured programs for each of those nouns. The development process can be achieved by mapping input/output screens which contain information displayed within the user interface. A collection of 'Signification Vectors' are used to generate programs in Lyee, each one relating to a specific requirement. Within each Signification Vector is a process of iteration, which establishes the program for that specific noun. The Lyee software methodology has already

been applied to many business applications. This work has used Lyee as an inspiration to continue this further in the multimedia context.

It is argued that it is normal to introduce a multimedia designer into the multimedia design process to aid the user, especially where the user is not expert in the development process. The primary concern of the work described here is to investigate the possibilities of reducing or removing the need for the multimedia designer in the process. The ultimate goal is to find ways in which the user or client can express their desire or intention directly to a computer system which can then produce a satisfactory multimedia product.

The role of the multimedia designer is to guide the user through the development process, listening to their ideas and advising on what is the most effective way of implementing the idea. If the idea was not viable then the multimedia designer would explain why so, and perhaps offer an alternative. The multimedia designer also advised on human-computer interaction issues in relation to the users ideas. From this process of interaction with the multimedia designer the user began to develop an understanding of some of the issues involved. Their intentions were observed to develop.

Within this multimedia CD-ROM production, it was the multimedia designer that produced the end product through various iterations, and with feedback from the user. As the system was built piece by piece, with feedback from the user, there was no observed misinterpretation about the users intentions, whereas in conventional software development this is a common problem.

If the multimedia designer was removed from this process then, on the face of it, it could be difficult for the user to produce a multimedia CD-ROM, especially if they are not familiar with the software that is used. However, users that were familiar with the software may be able to include all the material they wish into a CD-ROM, and create the 'look' and 'feel' that they require. On the other hand, they may not have the skill to organise the information in a logical way, or in a way that would be user friendly to a third party.

The issues raised in the above section are from the observations in the initial studies, and therefore may not relate to every case of multimedia development. The role of the multimedia designer within the process needs to be examined further. The team will produce a new multimedia CD-ROM for a new user, and with the help of an independent third party observer these issues will be explored.

2. The Studies

Studies have been made of the production of two multimedia CD-ROMS for two different artists. For one of the CD-ROMS the entire development process was carried out. The second CD-ROM involved making several modifications and additions to an existing CD-ROM. In both cases, the process has been considered in order to identify the issues that need to be investigated in order to consider the application of Lyee, or an extension of Lyee, in this area.

In order to produce the first CD-ROM, it was necessary to liaise with the artist to gain an insight into what they had planned for the CD-ROM. The first meeting was used to gain an understanding of the artists work, and to find out what material he wished to include on the CD-ROM. Once this was established a series of meetings were held on a weekly basis where the artist would be shown revised versions of the CD-ROM. This was also an opportunity for the artist to give feedback on what they had seen, and suggest improvements or changes that could be made. Throughout the development process the artist was keen that the presentation and feel of the CD-ROM would reflect his work.

The second CD-ROM needed to be developed from an existing CD-ROM, most parts of which were to be maintained. The new intension was an extension of the first CD-

ROM but the style and the layout had already been established, and therefore the amendments that were made needed to fit in with this. Within the sections that were to be changed, only part of the section would be amended before it was to be shown to the artist. This was so that the artist could see a sample of how the changes would look, and could provide feedback on whether it meet his expectations without the Multimedia designer doing unnecessary work. After a series of iterations the artist was happy with the changes, and these could then be applied to the other parts of the section.

Although both artists had an idea of what they wished to include on the CD-ROM, it was not always easy to express it in a logical and user-friendly way. In some cases it was necessary to map out the relationship between different screens, in order to show how the screens would appear to be linked to the end user. The result of this work, apart from the successful CD-ROMs, were better understanding of the process of moving from intention to software product in the case of a multimedia system.

2.1 Development of a CD-ROM: From start to finish

The first CD-ROM produced was a new project, and therefore the CD-ROM was developed from scratch. Before the first meeting was held, it was advised that the SE should familiarize themselves with the artists website. At this time the SE was unfamiliar ROM. From looking at the website the SE was asked to try and think about a concept that could be applied to the CD-ROM. The concept would determine certain characteristics of the look and feel of the CD-ROM. It was decided that the look and feel of the CD-ROM should reflect the work of the artist, and as such one of the artists works was used to form the main option buttons on the main menu screen. A similar colour scheme was used for the CD-ROM as on the website, because this suited some of the images that were to be included, and in order to maintain consistency with the website.

During the first meeting the artist indicated his desire for the CD-ROM to be split into two separate sections, each section containing different information. The artist was also keen for the CD-ROM to be easy to use, and that a user could pick up the CD-ROM and browse it for a few minutes, or could study the CD-ROM in greater detail. In order to allow this it was decided to include an 'Overview' section, which the user could browse to get a feel for the artists work, without going into greater detail in the other sections. It was during the first meeting that the artist indicated that they wished to include text, images and video into the CD-ROM.

Following the first meeting, a series of meetings were held weekly, where the SE would show the artist a revised version of part of the CD-ROM. This was where the artist could give feedback on what they saw, and suggest changes. Even when the SE was given all the material that was to be included within the CD-ROM, it was difficult to organize it in a logical way that was also set out in a 'user friendly' manner. The biggest difficulty for the SE was trying to get the artist to be specific about the navigational layout of the CD-ROM. At one stage the SE had to map out the navigational layout, to try and get the artist to be specific about what material should be contained in which parts of the CD-ROM. It was important to get feedback from the artist on this otherwise some material could be in the wrong parts of the CD-ROM.

The other significant issue in the development of the CD-ROM was relating to the design of individual screens that contain a lot of information (such as dealing with papers and their associated images). It was necessary to develop a consistency within the CD-ROM, so that individual screens had buttons in the same place, and that certain colours were used for certain parts of the screen. The final layout of these screens was developed gradually. Initially screens containing text (papers) and images would only contain these as raw images, but as the CD-ROM developed, it became necessary to include other items on

the screen such as navigation buttons, title bars, and colours. These too were to be kept consistent throughout the CD-ROM, and these additions were to fit in with the look and feel of existing material already on the screen. The placement of the first raw images was done considering the HCI (Human Computer Interaction) needs of the end user, and the navigation buttons, colours and titles were added later as the CD-ROM developed.

2.2 Development of a CD-ROM: Updating an existing CD-ROM

The second project involved updating an existing CD-ROM, where much of the content and design was to be maintained. The artist in this case knew which part of the CD-ROM they wanted to change, but they were not sure how to change it. The artist's initial idea of how to change this section of the CD-ROM was viable to implement, although after consulting the SE the drawbacks of that method were realized. The drawbacks were not related to the visual impact of the idea, but related to how 'friendly' the idea was to the end user from a HCI point of view. As a result the SE offered an alternative solution that the artist was happy with. At this stage the ideas were only conceptual, it was necessary to produce several prototypes so that the artist could see and get a feel for each of the alternatives.

A crucial step was in providing the artist with five alternative prototypes for him to choose from. These five could not represent all possible interpretations of his intention, but they were selected so as to cover the range as well as possible. The first observation to make is that the artist was not satisfied with reviewing the alternatives on his own. The situation was quite changed when the SE demonstrated them to him, however. Thus, human intervention even in a small way seemed to be important. During the demonstration the artist was able to select the preferred options but also to better articulate his thoughts about the CD-ROM design. Over the prototypes, a discussion took place in which specific concerns were mentioned and, with positive suggestions being made by the SE, resolutions found. The final decisions at the end of this session were more than a selection from the prototypes. They included variations on a given prototype. It was as if the demonstrations allowed the range of possibilities to be narrowed down and then the expression of more detailed intentions became easier. On the other hand, it may be that the intentions were only finally formed during this demonstration session. Discussion over prototypes was, in all probability, the trigger for clear thinking about intention.

2.3 Observations on the development process

When developing these multimedia systems it is important to try and understand the work of the artist, which is crucial for developing a concept that is used to develop the CD-ROM. It would be difficult to automate this part of the process due to the conceptual nature of the information that is needed.

From the two projects above it was found that the artist may not be aware of many of the HCI issues that are involved within the development process, which could mean that some of their ideas may not be viable or friendly to a third party user. It was also found that the artist is likely to have only a vague idea of how they hope the CD-ROM to look and feel, and it is up to the SE to help fill this gap with the aid of the artist. In the first case above it was often necessary to show the artist something in order to obtain feedback, this may be because it is difficult to imagine how something should look and feel without first seeing something which can be used as a comparison. This part of the process would also prove difficult to automate, not only because of the nature of the information, but also because this is an ongoing iterative process.

The iterative nature of the design process means that even specific screens may take several iterations before they are finalized. This is due to the fact that all of the screens are interrelated in the design process. This does not mean that they are related in the way that the end user navigates through the screens, but that if for instance a particular colour or font is used for a specific part of the screen, then this would also be used on other screens. This was important to maintain consistency throughout the CD-ROM.

It was observed in the projects above that the artists intention were developed as the CD-ROM was built, and with the guidance of the SE. Within the first project the artists developed more specific 'narrower' intentions throughout the development process. This was a result of the SE showing the artist various versions of parts of the CD throughout the development process. This process of refining the users intention relates to the work by Ohsuga, described in [5]. These more specific intentions would be sparked by something that the artist was shown, and the intention would be to modify something they saw. In the second project the SE advised the artist to change his intention because of the implications for the end user.

3. Using Lyee to develop multimedia systems

The Lyee software methodology has already been applied to many business applications, which are in many ways often different to multimedia systems. Within the development of business applications much of the information that is contained within the nouns and programs is quantitative. Such information could be a set of formulae or statements that determine the way the program will operate. The rules within the formulae would be structured so that every possible outcome could be accounted for. Here there is no ambiguity within the program.

The requirements for the development of a multimedia system are not always quantitative. Much of the information that is included in such a system is qualitative, such as images, sound and video. It is more difficult to express intentions about this sort of information in terms of nouns, which can then be generated into programs. It would appear that the implementation of Lyee may need to be extended in order to include some of these multimedia issues.

Within Lyee, programs are derived from nouns that come from users intentions. In straightforward cases, we may assume that the intention will not change, and that the initial intention is correct. Within the development of multimedia systems, however, the users intentions may change throughout the development process. For example, the user may have an initial intention that is not viable to implement, this could be due to their lack of expertise within the field of multimedia development. Thus the multimedia case leads to certain complex application issues in terms of the employment of the theory. Within multimedia system development the user often has broad intentions for what they hope to achieve, but these cannot easily be mapped directly into programs, because they are too broad. The definition of each of these intentions needs to become more specific, and the initial broad intention needs to be broken down into many smaller narrower intentions before a resulting program can be defined. The problem is to find a precise method for doing this. Achieving not only makes it easier to map into a program, as is normally done in Lyee, but also encourages the user to think more about the system they are hoping to achieve. Bearing in mind that the user may not be an expert in the development process, even if we were able to obtain a set of narrow intentions they may still be conceptual and difficult to map to a program because of the complexity of multimedia.

For multimedia systems the Lyee software implementation may require a new way of expressing users requirements and intentions. This would be to enable rich qualitative

requirements to be mapped into programs without losing sight of the initial requirement. The re-iteration of requirements also needs to be considered for circumstances when the requirements are changed by the user part way through the development process.

4. Key questions of multimedia systems development from intentions

As a result of the studies completed so far, a number of questions arise that will form the basis of the in-depth studies that must now be undertaken. Lyee is a methodology that enables us to produce source programs directly from an intention. The source program should express a user's inherently ambiguous intention as it is in a programming language. Within the development of multimedia systems, users' intentions are not always static, and the intention itself may be inappropriate for what the user requires. The initial intention may be ambiguous at the start of the process, but because of the nature of the intention it may be difficult to express it as a programming language. The intentions may need to be refined throughout the development process, where the user slowly becomes more aware of what they want, as they are able to see parts of the system as it is developed.

The systems that are developed such as multimedia CD-ROMs can start from a broad intention, where the user first considers the idea of creating a multimedia CD-ROM. This initial broad intention then becomes a larger set of narrower intentions, as the user begins to consider deeper issues regarding the multimedia CD-ROM. These more specific intentions would continue to be broken down until the user has an intention for every aspect of the multimedia CD-ROM. Some intentions may be qualitative because they contain an idea, or because the user is trying to create a certain 'feel' or 'mood' and these may prove particularly difficult.

The reason for the 'fuzzy' initial intention, that becomes a set of narrow refined intentions, is because users do not necessarily have an understanding of all the issues that are involved within the development process. Initially users may be unaware of the possibilities and limitations that are imposed on them. For example, users may have little awareness of human-computer interaction issues that are involved. The user may develop an awareness of these issues during the development process that may cause the initial intention to be refined. We will need to understand this process better. The method to be employed is first to understand the role of the multimedia designer/analyst in conventional development. We will then be in a better position to understand how, according to the Lyee theory, the engineer's role might be removed or minimised.

5. Possible implications for Lyee

Once the user has broken down their intentions, so that there is a specific intention for each specific part of the CD-ROM, this then can be mapped into a Lyee-based system, which would produce the CD-ROM the way that the user wants it. It is likely that the user will only be able to produce a detailed set of specific intentions part way through the development process, after a series of modifications. Lyee may need to be extended in order to cope with changing intentions.

The process of refining the intentions within multimedia CD-ROM development, at this time still requires the presence of a multimedia designer to aid the user. When the intentions are specific enough, then they could be mapped into a system to produce the end product. The intentions would need to be in a form similar to HTML, from which web pages are produced. It is likely that multimedia systems will need to be checked by the user and could be subject to a number of iterations.

The Lyee concept may need to be extended so that it can be applied to the development of multimedia systems such as CD-ROMs. This is because of the nature of the intentions. Often users intentions do not become clear until midway through the system and give feedback to the multimedia designer. If it were possible to map the refined intentions into some type of software generator, it would be doing the same task as the multimedia designer has been doing for the previous iterations, from which the user intentions are developed further and refined. Therefore it is concluded the study of the process and the role of the multimedia designer will form a firm basis for continuing the study and development of a multimedia application understanding of Lyee.

6. Conclusions

From the studies carried out in the two projects it was found that the SE was crucial to the development process of the multimedia CD-ROMS, and as such it would be difficult to remove the SE from the complete process. It would be difficult to remove the SE from the traditional requirement specification phase, because this is an ongoing process, and the specification may not be clear at the outset. The development of multimedia systems appears to be an iterative process where the system is developed with feedback from the person who desires the system. In order to understand this process further it is necessary to develop a process model. The spiral model of software development [6], introduces the idea of prototype systems being developed throughout the whole process, with the idea that this makes it easier to understand the requirements for the next stage of development. This model uses the main stages of software development, but shows that several iterations are required before the final version is complete. This would be an appropriate model to adapt because it uses the same stages of software development that are present in the development of multimedia systems. The possibility of developing a knowledge based system, a variation of the 'Programmers Apprentice' [7], could be explored further, although this may prove problematic because of the infinite number of ways that a multimedia system can be formed.

It is proposed to investigate extending Lyee so that it can be successfully applied to the production of multimedia CD-ROMS, as mentioned above. The iterative nature of the design process may need to be addressed, together with the changing user intention that is developed throughout the process. A key factor that was observed in the studies mentioned above was that the artist was able to give feedback whenever shown something by the SE. This too may need to be taken into consideration because it provides the spark from which intentions form and develop.

Although the users intention is refined throughout the process, it remains an open question of how to formalize the confirmation process of whether the users intention has been successfully reflected in the final iteration of the multimedia CD-ROM. Here there was no criteria or performance model to judge this, and from the studies carried out it was found that this was difficult to measure this, especially in circumstances where the artists intention had to be refined to take into account HCI issues.

Acknowledgements

Thanks are due to Ernest Edmonds for his supervision and help with this work. Dr. Hamido Fujita has made valuable comments on the work and the author would like to also thank him, together with Mr Negoro and the Lyee team for making the study possible.

References

- [1] Little, P., 'Project management and management of design: Teaching and tools', (1998), *Artificial Intelligence for Engineering Design, Analysis & Manufacturing*, 12, 49-50
- [2] Dearden A.M. and Harrison M.D., (1997), 'Abstract models for HCI', *Int J. Human- Computer Studies* 46, 151-177
- [3] <https://www.lyee.co.jp/webapp/en/tokkyo/index.htm>
- [4] Negoro F., (2000) 'Principles of Lyee Software', *Proceedings of 2000 International Conference of Information Society in the 21centurey*.IS2000,Japan 2000.
- [5] Ohsuga S., 'How can a knowledge-based systems solve large-scale problems?: modal-based decomposition and problem solving', *Knowledge-Based Systems*, Vol 6, No. 1, March 1993
- [6] Boehm B., (1998). 'The spiral model of software development and enhancement', *IEEE Computer*, 21 (5), 61-72
- [7] Waters R.C., (1982), 'The programmers apprentice: knowledge based program editing', *IEEE Transactions on Software Engineering*, Vol SE-8, No. 1

Capturing Collective Intentionality in Software Development

Benkt WANGLER, Anne PERSSON

*Department of Computer Science, University of Skövde,
P.O. Box 408, SE-541 28 Skövde, Sweden*

Abstract. This paper discusses the elicitation of intentions as perceived by the Lyee methodology. We suggest a participative approach to developing an intentional model, called the collective intentional model, which results from reconciling the views of a diverse set of individual stakeholders.

1. Introduction

The aim of the Lyee methodology [1] is to bridge the gap between the intention of the stakeholders and the software product to be developed by means of an orderly and tool supported way of working based on the principles of scenario function programming as defined in [2]. However, since there are usually several stakeholders there is not one single intention, but every single stakeholder may have his own wishes and desires. Quite so often the intentions are also in conflict with each other. This paper presents an approach to overcome such inconsistencies and to reconcile different stakeholder intentions.

The approach presented in this paper complements work by Ekenberg and Johannesson in that it provides the input ((2) in Figure 1) to the mapping ((3) in Figure 1) described in [3].

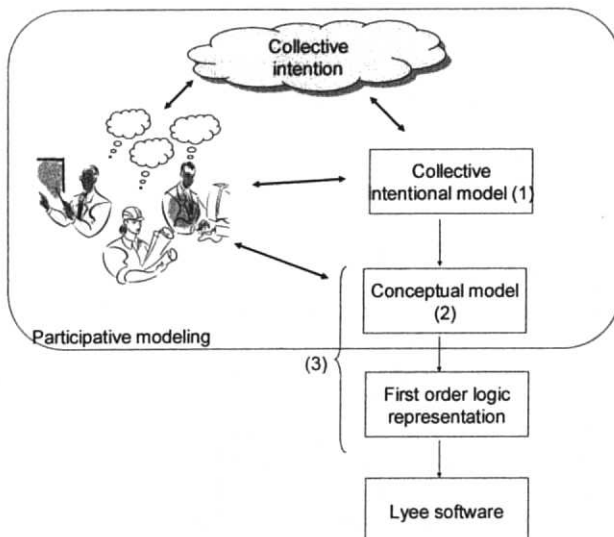


Figure 1. The process of transforming the collective intention to Lyee software.

The collective or enterprise intention is discussed in some more detail in Section 2 of this paper. The collective intentional model (1) is comprised of

- a means-ends (goal) model
- a concepts model, i.e. a model of the 'things' dealt with in the enterprise
- a process model

These models would then form the basis for constructing the UML class and state diagrams that make up the conceptual model (2), which in turn serves as the input to the mapping (3) to first order logic. For details of this mapping, the reader is referred to [3].

The remainder of the paper is organised as follows. Section 2 defines the notion of a collective intention. In Section 3 it is proposed a participative approach to discover the collective intention. Section 4 discusses the preconditions of succeeding with such an approach, while Section 5 describes the competency requirements. Finally, some concluding remarks are given in Section 6.

2. The collective intention

Goal-driven software requirements capture builds on the notion of software requirements being developed from the goals of various business stakeholders [4]. The Lyee software methodology is based on a similar notion, in that it captures the intention or goal of the customer and from that builds the software in question. However, capturing and describing this intention is not an easy matter, since the customer may not be completely aware of its needs. Furthermore, the customer's wishes are sometimes in conflict with each other in that some may not be easy and perhaps not even possible to implement at the same time.

In the sequel we will describe our ideas further, building on the assumption that the notion of intention takes many forms in an organisation.

1. Every enterprise has a *mission*, often but not always explicitly expressed in a mission statement.
2. The enterprise states (should state) *goals* vis-à-vis that mission. The goals form a hierarchy (directed, non-cyclic graph) of objectives on different levels of abstraction. The lower level, more specific, goals constitute the realisation (implementation) of the higher-level goals. We sometimes refer to those, more specific, objectives as *business rules*. Low-level goals may also be considered as *tasks* to be carried out in order to achieve higher-level goals.
3. Goals may be conflicting, i.e. what is stated by one goal may not be possible to realise at the same time as some other goal.
4. *Strategy* and *policies* are formed to define the way in which goals are to be accomplished, and should be formulated such that conflicting goals are somehow relaxed. The strategy and policies may be considered as the more specific, lower level parts, of the goal hierarchy.
5. The mission statement, goal structure, strategy and policies together form the *enterprise intention*.
6. An organisational structure is formed such that organisational units on different levels are made responsible for various goals and tasks.
7. Organisational units comprise sets of individual actors. Actors are usually human but may be artefacts such as computerized information systems. The requirements for such systems are derived from the objectives the organisational unit has to achieve.
8. The union of the consciousnesses of each human actor may be said to constitute the organisational (unit) *collective consciousness*.
9. The intention of individual actors vis-à-vis the organisation should reflect the intention of the enterprise.

10. Intentions of individual human actors may be conflicting. A *common (collective) intention* may be formulated in consensus-creating activities such as enterprise modelling (e.g. goals, concepts and process modelling).

The collective intention, hence, incorporates both the enterprise intention and the intentions of the involved actors. In the following, we will outline a way of working that will help in capturing and negotiating the enterprise intention.

3. Discovering the collective intention

Enterprise Modelling (EM) is commonly used in the early stages of software development [5, 6]. In particular, software development practitioners claim that EM is effective for gathering business needs and high-level requirements [7], i.e. for specifying the intentions of the stakeholders of the system.

Any method for EM has two main components:

- 1) A meta-model, which defines the modelling constructs, syntax, semantics and graphical notation used to create an Enterprise Model. This is the modelling language.
- 2) A suggested process for creating an Enterprise Model. During this process different ways of working are applied in order to elicit and develop the knowledge of business stakeholders or domain experts. Typical examples of ways of working are facilitated group sessions and interviews.

This paper focuses on the second component, the process of discovering the intentions of the stakeholders. We argue that a participative way of working is the most suitable approach to this end. In the participative approach to EM, the stakeholders in question collaboratively develop Enterprise Models in facilitated group sessions. This type of participation is *consensus-driven* in the sense that it is the stakeholders who “own” the model and hence decide on its contents. In contrast, *consultative* participation means that analysts create models and that stakeholders are then consulted in order to validate the models. In the following, the participative approach to EM will be described and discussed.

3.1 Overview of the participative modelling process

In the participative approach to Enterprise Modelling, stakeholders meet in modelling sessions, led by a facilitator, to create models collaboratively. During the modelling sessions, models are documented on large plastic sheets using paper cards. The “plastic wall” is viewed as the official “minutes” of the session, for which every participant is responsible.

Setting up and carrying out modelling sessions using a participative approach to Enterprise Modelling can briefly be described as follows [8]:

- 1) Before the first modelling session the domain experts/participants are selected and interviewed.
- 2) The modelling session is then prepared. It is critical that this preparation is thorough. Objectives are defined and a schedule for the session is prepared.
- 3) The modelling session is carried out following the prepared schedule. During the session the facilitator is a crucial actor as manager of the group communication process and as keeper of the method knowledge.
- 4) After the modelling session, the models are documented by using some computer tool and a walk-through session is carried out with the participants in order to validate the models.
- 5) Based on the documented models, new modelling sessions are prepared and carried out until the problem at hand has been sufficiently analysed.

The required time to prepare a PEM activity should not be underestimated [7]. Our research has shown that spending a great deal of time on preparation is well worth the effort in the long run. In fact 43 % of the time spent on a PEM activity should be focused towards

preparation. In particular, preparatory interviews should always be made with the modelling participants before the first modelling session.

3.2 Arguments for adopting a participative approach to Enterprise Modelling

The importance of active user participation in systems development has been emphasised for a number of years, by practitioners as well as by researchers. The idea behind this movement is that the technical quality of a computer-based information system is of no significance if its users are not satisfied with it and consider it to be useful for carrying out their tasks. Participative Design (PD) has in fact developed to be a research topic in its own right (See further e.g. [9]).

There are at least three reasons why a participative approach to modelling should be used, if possible [7]:

- The quality of the Enterprise Model is enhanced.
- Consensus is enhanced.
- Achievement of acceptance and commitment is facilitated.

The *quality of a model is enhanced* if created in collaboration between stakeholders, rather than resulting from a consultant's interpretation of traditional interviews. From a practitioners' point of view a "good" model is a model that makes sense as a whole and that is possible to implement as a solution to some problem. The participative process capitalises on the fact that when a group of people discuss a problem they all associate their own thinking with the stated views of other people in the group and this enhances their own thinking. It also improves the knowledge of the analyst compared to traditional interviewing techniques since the discussions between stakeholders give a richer view of the problem than talking to the stakeholders one by one. This means that ideas are validated and improved when a group works with them. In fact, people in organisations, especially large ones, do not talk to each other about their work as much as one may think [7]. In the situation where the analyst is not familiar with the problem domain and interviewing techniques are used, the holistic view of the problem is therefore at risk. Hence, the adoption of a participative approach can be seen as a quality assurance measure.

Consensus is enhanced when a participative approach is adopted. If there are different views in a group and the group is collectively responsible for the resulting model it seems that the group has two choices. Either it strives to achieve consensus between the participants or an open conflict is established that will need to be resolved through negotiation. Either way, a common view will need to be accomplished.

The adoption of a participative approach to modelling actively involves stakeholders in the decision making process, which facilitates the *achievement of acceptance and commitment*. This is particularly important if the deployment of a new piece of software involves changing the current work practices of the stakeholders. In fact, if a stakeholder has *considerable* knowledge concerning the needed functionality of a proposed information system, the chances of her/him accepting the system increases if she/he is given the opportunity to actively contribute this knowledge [10].

4. Preconditions for successful application of the approach

To achieve the desired effects of the participative approach to modelling, the following preconditions are the most critical [7]:

- 1) *The stakeholders who are most suited to participate in the modelling sessions must be allowed to do so.* This means that the most suitable are chosen and that they are given time to actually attend the sessions. The modelling group should consist of stakeholders that represent different views of the problem at hand so that the full scope of the problem domain is covered. Each participant should find the problem important enough to

contribute actively to motivate commitment and active participation in solving it. See further Section 5.1.

- 2) *The involved domain stakeholders accept the method.* The modelling participants need to accept the approach and understand its main ideas. In an authoritative culture, the idea of participation and openness probably does not appeal to the modelling participants. See further item 3. However, even in a consensus-oriented culture, some aspects of the group modelling technique may seem strange and awkward to the modelling participants. This may cause them not to contribute as much as desired.
- 3) *The organisational culture is consensus-oriented.* In this paper a consensus oriented culture is defined as a culture where subordinates can question their managers, the dialogue between levels of the organisation is open and direct and reward systems encourage initiatives from all levels of the organisation. Using a participative approach in the “wrong” organisational culture is likely to cause modelling activity to fail. It might also bring out conflicts between people that are difficult to mend afterwards. This makes organisational culture one of the most critical issues in the application of the participative approach to modelling.
- 4) *The modelling participants and the method provider must have the authority to act freely during the project.* The effectiveness of the participative approach relies heavily on the authority of involved actors, especially the method provider and the modelling participants. Important decisions will be taken in the modelling sessions and therefore the participants need to know that they are authorised to take these decisions or are authorised to have an influencing opinion regarding the topic in question. There are two apparent risks involved. If the method provider is not properly authorised she/he will not be able to carry out the work according the participative approach and she/he cannot guarantee the quality of the modelling result. If the participants do not feel that they are properly authorised they will not be motivated to actively contribute their knowledge to the modelling session.
- 5) *A constellation of method providers with skills matching the project definition is assigned to the project.* The competency of the method provider is a critical resource in a PEM project. A high degree of problem complexity requires a truly experienced method provider team leader. The size of the project defines the need for co-ordination efforts. A large modelling project will need an experienced and skilled method provider team leader with a holistic view. Designing the future state or radically changing the current state requires a skilled facilitator in the modelling sessions than describing the current state. See further Section 5.2.

This line of reasoning boils down to one critical issue: It is critical to the success of participative Enterprise Modelling to involve the “right” people (on the customer and method provider sides) in terms of knowledge, skills, and willingness to contribute.

It is equally critical that they can act in an atmosphere, which is characterised by openness and confidence between all stakeholders involved. This is at the heart of the idea of the participative approach to modelling. If an atmosphere of openness and confidence cannot be created, the modelling participants will most likely hold back their knowledge contribution because of various fears. Modelling can be carried out but the effects of the participative approach as described above will not appear. Hidden agendas, lack of management support and low acceptance for the way of working are also potential risk factors in a participative modelling project. More specifically, they seem to be risk factors for the availability of required competency. The risks pertaining to these factors can be summarised as follows [7]:

- 1) Poor management support for the project may cause the modelling participants to feel less motivated to commit their time and effort to the project, which in turn is likely to cause poor model quality. The competency pertinent to the individual modelling sessions will

hence not be physically available. Also, poor management support most likely will result in the modelling participants being less authorised to actually influence the outcome of modelling. Finally, poor management support causes unwillingness from the customer to assign resources in terms of time and money. This means that modelling participants are not given enough time to participate in the project, causing the needed competency to be physically unavailable, in turn causing poor model quality.

- 2) Hidden agendas may cause unwillingness among the modelling participants to contribute their knowledge, given that they know or suspect that such agendas exist. Hidden agendas may also cause restrictions with regard to the composition of the modelling group. The customer might choose to involve participants with politically correct opinions supporting the hidden agenda although they are not the optimal choice in terms of domain knowledge.
- 3) If acceptance for the EM method suggested by the method provider is poor among the modelling participants they will most likely not commit their best effort to the modelling sessions, which will in turn cause poor model quality. If acceptance for the EM method is poor on the part of the customer, she/he will not give his open support for it. This may cause the modelling participants to mistrust or not accept the suggested method and hence not commit their effort.
- 4) If the opinions and ideas of modelling participants are not given the proper authority and consideration in the development process, the modelling sessions will most likely produce a result that is not used. This means that modelling has no real effect on the problem at hand and hence the resources spent are wasted. If a modelling group consists only of the “politically correct” participants, the needed competency may not be represented in the modelling group (See further item 2. above) Furthermore, if the modelling participants *feel* that they are not authorised, they will be reluctant to contribute their domain knowledge in the modelling sessions.

In summary, the adoption of a participative approach to modelling should not be taken lightly. It is more demanding on the method provider team as one may think at first. The competency requirements for participative modelling are discussed in the following.

5. Required competency in the participative modelling process

The main actors in the participative modelling process fall into two main categories: *domain experts* and *method providers*. The competency of domain experts and the method providers are most crucial resources in an EM project.

An Enterprise Model comprises knowledge regarding different aspects of some organisation. Domain experts provide and develop this knowledge in modelling sessions. To create the Enterprise Models, using a participative approach, a *method provider* is needed. The method provider has the necessary knowledge with regard to the EM method used and the participative approach to modelling.

The *customer* is the main responsible party for the modelling project on the domain expert side. In the modelling sessions, the *modelling groups* consist of *modelling participants*, which are domain experts concerned with the problem at hand. These domain experts may also be involved in other projects that are related to the one being planned. The customer may also participate in modelling groups. The main problem being addressed has a *problem owner*, which mostly is the customer. For sub-problems, there may be also other problem owners, e.g. other modelling participants.

The method provider negotiates and plans the project together with the customer. A *facilitator* moderates each modelling session. In a session there can be more than one facilitator. A larger modelling project will typically have several facilitators forming a *method provider team*, which is headed by a *method provider team leader*. The team leader is often an

experienced facilitator. The main responsibility of the method provider is that the chosen EM method is appropriately used and that the project resources are used in a way that enables the project to be completed on time and in such a way that the project goals are achieved. The main responsibility of the domain experts is towards solving the actual problem at hand using their domain knowledge. It is essential that this distinction be made and that it is made clear that the domain experts are responsible for the knowledge content of the models.

5.1 *The competency of the domain experts*

The domain experts need to have knowledge that is pertinent to the problem that the modelling session addresses. Otherwise they will have no possibility to contribute an opinion on the subject matter. In addition to knowledge of the problem addressed, some personal characteristics are also desired in an “ideal” modelling participant, such as the ability to abstract and generalise, social skills, ability to verbalise and communicate, creativity etc. One of the most important requirements for an individual domain expert is the motivation to participate and to contribute to solving the problem at hand. This can make up for lack of some personal characteristics.

The composition of the modelling group is crucial to the achievement of the goals for the modelling session. One aspect to consider is that different areas of knowledge are represented in the modelling group so that the scope of the problem at hand is covered.

The “direction” of the analysis, i.e. if the analysis concerns the current state of affairs or the future state, also defines requirements for the group’s composition. It is often the case that the people that are deeply involved in the problem can describe the current state. However, when it comes to the future, a different type of stakeholder is needed. Here we need the visionaries, perhaps those that represent a higher level of the organisation and see more than just the little piece of the puzzle.

Another aspect of group composition is the number of participants in a group. It seems that an ideal number is between 5 and 10. If the group is too large it is difficult for the facilitator to manage the group process, but a really small group can also influence negatively on the modelling result. For further details about the dynamics of a modelling group, see e.g. [11, 5, 12, 13].

5.2 *The competency of the method providers*

The competency of the method provider is a critical resource, being responsible for the effective adoption of the chosen approach and for the project reaching its goals using the assigned resources.

There are three main levels of general method provider competency (Figure 2).

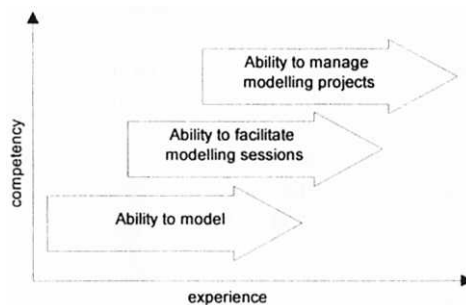


Figure 2: *Three levels of method provider competency*

The *first* and most basic level is that the method provider has the ability to model, i.e. to choose a suitable formalism for the problem at hand, to use the modelling method's meta-model in order to represent some chunk of knowledge. Furthermore, representing the knowledge in such a way that the model actually reflects that knowledge.

The *second* level of competency focuses on the ability to facilitate a modelling process, i.e. to lead/facilitate a modelling group in the process of collaboratively creating an Enterprise Model. This ability is very much based on knowledge about the effects of modelling, the principles for human communication and socialisation, especially in groups, as well as the conditions of human learning and problem solving (cognition) [12].

Facilitation is a general technique used in group processes for a wide variety of purposes¹, also within modelling. In facilitation we include the activities of preparing modelling sessions, facilitating them and documenting the result.

An important part of the competency of a modelling facilitator is "social skills". These "social skills" are personal characteristics, which can be further described as follows. [12]:

- *Listening skills.* Listening is not only about listening to what is actually being said. Listening behind the words for what is really meant is the important aspect of this skill.
- *Group management and pedagogical skills.* The leader role is facilitated by the ability to motivate and keeping the modelling facilitators interested. The ability to detect and solving potential conflicts is also part of this skill.
- *Act as an authority.* To be an authority in this context is not the same as being authoritative. To act as an authority is to create trust for your own competency and to make the modelling participants feel that you know what you are talking about.
- *Courage and ability to improvise.* In a modelling session situations constantly arise that call for improvisation and courage to act beyond what has been planned.

It is often an advantage to have more than one facilitator in a modelling session. Some facilitator teams share the work so that one person documents and one facilitates. Others take turns facilitating the group. However, the team relationship must be completely equal in order to function properly. Every member of the facilitator team must be considered by the group to be an authority.

The *third* and top level emphasises the ability to co-ordinate and lead modelling projects, involving a team of method providers, toward achieving project goals. Leading a modelling project involves negotiating the project as well as monitoring it and reporting the results. If facilitation of modelling sessions can be considered to be the micro process then the whole project is the macro process. The macro process, especially in large projects, requires a great deal of experience. Of course the usual project management skills are needed, but apart from that the macro level requires need the ability to see how the whole set of modelling sessions and their results hold together and how they contribute to achieving the project goals.

A critical task in leading a project is negotiating the project definition and resources and also judging which approach to modelling that is appropriate in that situation. It is advisable that an experienced method provider negotiate the project the less experienced focus on facilitating modelling sessions.

6. Concluding remarks

In the previous sections, we have argued that the participative approach is a useful way to discover the collective intention. Having done this, finally arriving at UML models, the models can then be used as input to the transformation suggested by Ekenberg and

¹ See International Association for Facilitators (IAF) <http://iaf-world.org/iaflinks.htm> (as is 2001-08-30)

Johannesson [3]. It deserves pointing out, again, that we usually apply our approach in two steps so as first to arrive at a preliminary model consisting of

- a goal model revealing the goals and objectives of the organisation in the form of a means-ends hierarchy,
- a process model outlining the broad activities to be conducted in order to achieve (some of) these goals and the various relationships between these activities, and
- a concepts model defining the “things” dealt with in the organisation as well as relationships between these “things”.

From these models we develop the UML class and state diagrams ((1) in Figure 1) that may then be used as input to the transformation to first order logic.

We strongly believe and we do have some evidence from practice [5,7,14] that the approach accounted for in this paper is in fact the best way to alleviate the obstacles by the fact that different stakeholders may have differing views on information systems requirements.

References

- [1] Negoro F., “Principle of Lyee Software”, International Conference on Information Society in the 21st Century (IS2000), Tokyo, 2000.
- [2] Negoro F., “Intent Operationalisation for Source Code Generation”, Proceedings of SCI 2001, Orlando (FL) USA, 2001.
- [3] Ekenberg L., Johannesson P., “UML as a first order transition logic”, in Proceedings of 12th European-Japanese Conference on Information Modeling and Knowledge Bases, Krippen, Swiss Saxony, May 2002.
- [4] Bubenko, J. A., jr and B. Wangler, “Objectives Driven Capture of Business Rules and of Information System Requirements”, IEEE Systems Man and Cybernetics '93 Conference, Le Touquet, France, 1993.
- [5] Nilsson, A. G., Tolis, C., and Nellborn, C. (eds.), “Perspectives on Business Modelling: Understanding and Changing Organisations”, Springer-Verlag, 1999.
- [6] Bubenko Jr., J. A. and Kirikova, M., “Improving the Quality of Requirements Specifications by Enterprise Modelling”, in Nilsson, A. G., Tolis, C. And Nellborn, C. (eds.), Perspectives on Business Modelling: Understanding and Changing Organisations, Springer-Verlag, 1999.
- [7] Persson, A., “Enterprise Modelling in Practice: Situational Factors and their Influence on Adopting a Participative Approach”, PhD Thesis, Department of Computer and Systems Sciences, Stockholm University/Royal Institute of Technology, Sweden, ISSN 1101-8526, 2001.
- [8] Bubenko, J. A. jr., Persson A. and Stirna, J., “User Guide of the Knowledge Management Approach Using Enterprise Knowledge Patterns”, HyperKnowledge project deliverable, project no IST-2000-28401. Dept. of Computer and Systems Sciences, Royal Institute of Technology, Stockholm, Sweden, 2001.
- [9] Schuler, D. and Namioka, A. (eds.), “Participatory Design: Principles and Practices”. Lawrence Erlbaum Associates, Publishers, 1993.
- [10] Saleem, N., “An Empirical Test of the Contingency Approach to User Participation in Information Systems Development”, Journal of Management Information Systems, Volume 13, No 1, pp 145-166. M.E. Sharpe Inc., 1996.
- [11] Willars, H. et al, “TRIAD Modelleringshandboken N 10:1-6” (in Swedish), SISU, Electrum 212, 164 40 Kista, Sweden, 1993.
- [12] Astrakan Strategisk Utbildning AB, “Högre kurs i modelleringsledning” (In Swedish), Course notes Version 1.1, Stockholm, Sweden, 2001.
- [13] Bergquist, S. and Eide, H., “Team Games –snabbaste vägen mot högpresterande arbetsprocesser” (in Swedish), Frontec AB, Sweden, 1999.
- [14] Persson, A. and Stirna, J., “An explorative study into the influence of business goals on the practical use of Enterprise Modelling methods and tools”, In Tenth International Conference on Information Systems Development (ISD2001), Royal Holloway, University of London, 5-7 September 2001.

Chapter 6

Enterprise Software Models and Software Engineering

This page intentionally left blank

Extending Lyee Methodology using the Enterprise Modelling Approach

Remigijus GUSTAS and Prima GUSTIENÈ
Information System Department, Karlstad University, Sweden

Abstract. Most software methodologies are heavily centred on system development issues at the implementation level. Such methodologies are restrictive in a way that a supporting technical system specification can not be motivated or justified in the context of organizational process models. A blueprint of enterprise infrastructure provides a basis for the organization's information technology planning. It is sometimes referred as enterprise architecture. The major effort of this paper is the demonstration of the Lyee software engineering methodology extensions by using the enterprise modelling approach. Integration of the pragmatic, semantic and logical software requirements by using the enterprise models is the main issue of the presented research.

1. Introduction

Nowadays the organisational systems must adopt to fast changing conditions of environment in order to survive. Thus, they are in the process of permanent change. One of the common software development failures is that, by the time the information system application is produced, quite often it is no longer relevant to the existing work practices [26]. Software system problems very often occur for a simple reason that various applications do not fit or they can not support the organisational components as it was anticipated. There are some important attempts to overcome the software component fitness and integration problems by designing software in a non traditional way. The Lyee methodology [14] is a way for building software on a basis of various logical structures of layouts by which information system users are supported in performing their tasks.

Companies and institutions in the public sector can be regarded as organizational systems that are supported by the computerised information systems. If an organizational activity could be unambiguously conceptualized, then it would determine the appropriate communication patterns in which a supporting information system can be useful. A supporting system cannot be defined until a definition of the supported system is available [5]. Thus, information system analysis process must first concentrate on modelling of the organizational system (supported activity), which the computerized part of information system is to serve [22].

Many customer organisations have information technology (IT) support in their companies. The IT experts are educated enough to present various screen, printout and data file layouts to be implemented. Nevertheless, in many cases, these logical design structures, like data structures in the process of relational database design, can not be identified without a complete understanding of how an overall information system is working. A long list of software system development failures demonstrates that in many cases such understanding is not available. Furthermore, the customers have great difficulties to capture

and to communicate unambiguously [11] information system requirements by using natural language.

Software maintenance problems arise for a reason that every enterprise has to survive a systematic change at a time new software is introduced. Various studies of change management problems in different companies and in public sector have demonstrated that the explicit representation of the organisational and technical system infrastructure is a necessary condition to understand orderly transformations of the existing work practices. If initial software requirements are presented without taking into consideration a context of the organisational environment in which a technical system is operating, then the requirements engineering approach can not be considered as viable, because it is not able to address software quality and IT fitness issues.

A blueprint of enterprise infrastructure provides a basis for the organization's information technology planning. It is sometimes referred as enterprise architecture [26], [25]. Just as the complex buildings or machines require explicit representations of their design structures, so does an invisible enterprise infrastructure. It needs to be captured, visualised and agreed to maintain orderly transformations. There is no other way to study the organizational and technical process fitness when the new software components are introduced, without describing an infrastructure of the existing system prior to the designing a new one.

Most software methodologies are heavily centred on a specification of the technical system part. Thus, they are restrictive in a way that a supporting system specification can not be motivated or justified in the context of organizational process models. The consequence is that to apply these approaches in some areas such as electronic commerce is not so simple. One of the problems is that business processes are spanning across organisational and technical system boundaries. These boundaries are not always clear. They are changing over time. If boundaries are not clearly identified and mutually agreed, then the result would be a misunderstanding between system users and designers.

The paper is organised as follows. In the next section some opportunities and limitations of the Lyee approach are given. It advocates the necessity of model based approach. The enterprise modelling approach is shortly introduced in the third and the fourth section. The conclusion section outlines perspectives of an extended approach and the future work.

2. Opportunities and Limitations of the Lyee Approach

A starting point of the Lyee requirement engineering process is words of the natural language sentences, which are used to make decisions on various screen, message, printout and file layout structures. The logical layout structures are used as a basis for definition of the Process Route Diagrams [14]. Process route diagrams are used in the input of LyeeAll case tool, which is able to create a source code of a specific software application. The logical layouts and derivation rules (application logics) of specific concepts are coded by using the conventional programming languages. The control logics of software component are created by the LyeeAll according to a process route diagram.

The most important contribution of Lyee approach is that an internal application control is separated from coding of the logical design and concept derivation rules. In the conventional software development, integrity among all these three parts is responsibility of a designer and programmer, that is why software testing plays an important role in an overall development life cycle. By using Lyee approach, the internal application control is materialised through the scenario function [14]. It is derived automatically according to the process route diagram. This is illustrated by Figure 1.

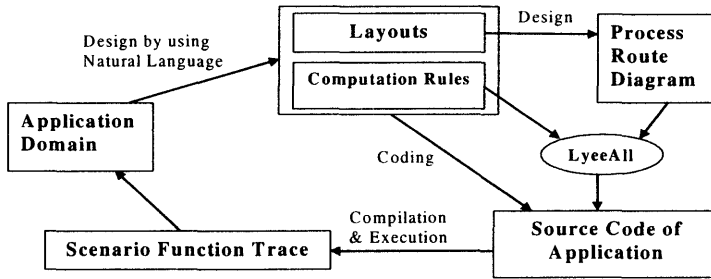


Figure 1. Core structure of the Lyee methodology

The weakest among all presented links in the core structure of the methodology is the way of how various rules and logical layout structures are captured. The logical design structures are formulated according to users' requirements by using a natural language. Ambiguities, incompleteness and inconsistencies that may take place among logical layout structures are difficult to identify and resolve for a reason that an integrated view to the software system and organisational system infrastructure is not available. It may cause difficult decision problems on how various layout structures must be shaped.

Misunderstanding between information system users and designers is a common problem. A precondition of a successful requirement engineering process is a mutual understanding. It can not be achieved without close cooperation of both parties. The problem is that the users do not want to be the software engineering experts. On the other hand, the designers have difficulties to understand the organizational system requirements in which a supporting software system is supposed to operate. According to the enterprise modelling approach, one of the problems in most of conventional software development approaches is that the implementation level requirements can not be represented without taking into consideration the organisational requirements, because they are not able to address some extremely important issues of software quality and IT fitness.

Every enterprise has to survive a systematic change at a time when the new software is introduced. Various studies of change management problems in different companies have demonstrated that the explicit representation of the business infrastructure is a necessary condition to understand orderly transformations of their work practices. The Lyee approach is built on the assumption that software developers are provided with logical structures of layouts by which information system users are supported in performing their tasks. In many cases information system users are not able to present technical system requirements in this way for several reasons. They need to be guided to achieve a mutual agreement on how the logical design of the screen or file layouts should look like. Such an agreement can not be reached without a complete understanding of how overall information system is working. A long list of software system development failures demonstrates that the assumption that information system users are IT experts is not correct. Most customers are not able to describe precisely requirements on how a design of the screen and file logical layout structure should look like.

One of the most important problems of software engineering is to maintain the existing software. This problem is due to the fact that software development methods have adopted a purely process-driven or purely data-driven approach. Many powerful modelling techniques [3], [23] do not provide any guidance on how to control the integrity and consistency of specifications that are presented on various levels of abstraction. As a consequence, the quality and reliability of the software development suffers significantly.

Another big problem is the implementation bias of many information system modelling techniques. The same concepts have been applied to the design and analysis stages, without

rethinking these concepts fundamentally. Enterprise modelling and integration should deal with the conceptually relevant aspects and it can not be influenced by the possible solutions to the problem. Many methodologies lack a systematic approach to the assessment of software quality and change management. The idea of model-driven approach [18] is to provide solutions for solving such problems. Model-driven software development follows the idea, pioneered by Jackson, that the complexity of information system is driven by the complexity of the underlying world [12]. By analysing and modelling a so called enterprise infrastructure, not concentrating on the specification of the desired functionality, information system developers are able to better manage the complexity of developed system.

The main focus of this paper is on demonstration of the advantages of an extended Lyee methodology by using the enterprise modelling approach. The enterprise modelling language can serve as a starting point for graphical representation of various users' intentions all the way across the organisational and software system boundaries. The outcome of the enterprise modelling process can be defined in terms of components and their interfaces with the structural definition of the printout, message, screen and file layouts. Various layouts can be viewed as elements of the actor oriented diagrams that define communication patterns among the collaborating actors [24] involved. Enterprise models suggest a new way of defining the enterprise infrastructures. It is based on an intentional way of thinking and on a new way of reasoning.

An extended approach should take shape of what the architects used to do when they construct buildings. The graphical representations are used for the purpose of visualisation and reasoning about information system quality. Graphical schemas of the objectified intentions may help us to resolve some information system inconsistency and integrity issues even before the layouts of screens, messages, printouts and files are submitted as an input of LyeeAll case tool. This is represented by Figure 2.

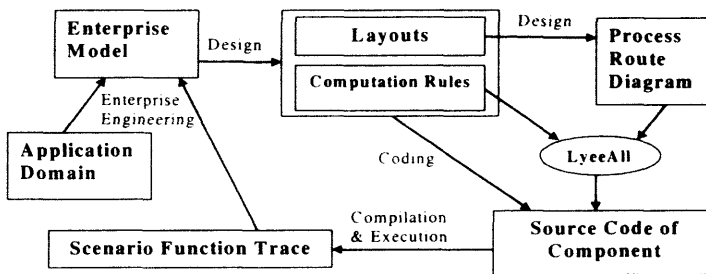


Figure 2. Core structure of the extended Lyee methodology by using the Enterprise Modelling Approach

Software systems should support business processes and software, to be regarded as a value-added technology, must fit various business processes. Change management in the organisational part or in the technical (software) part of the system is a big challenge, because even a simple deviation from the traditional business practice may be considered as a symptom for a new problem. The key issue is determination of the true IT needs and how these needs are integrated into the overall organisational system.

3. Enterprise Modelling and LYEE approach

Enterprise engineering is a branch of requirements engineering that deals with the information system modelling and integration [20]. At the same time it can be viewed as an extension and generalisation of the system analysis and design phase. Thus, enterprise

modelling takes place during the early, middle and late information system development life cycle. The most difficult part of the enterprise modelling is to arrive at a coherent, complete and consistent graphical description [10] of a new information system. The concept of enterprise here should be interpreted in a very wide sense [4]. It is actually aiming for description of pragmatic, semantic and syntactic aspects of the whole information system or some part of it.

The concept of enterprise in the context of information system development denotes a limited area of activity in the organisation, which is of interest by a system analysis expert. A number of methods in the system engineering methodology are directly concerned with a process of requirements engineering [15]. The ultimate goal of the enterprise modelling is to introduce a common basis for integration of various dependencies [11] that are used in requirements engineering and conceptual modelling [16] at the syntactic, semantic [19] and pragmatic level. This idea is illustrated in Figure 3.

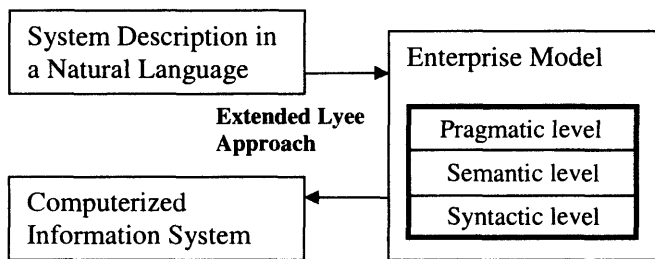


Figure 3. Extended Lyee approach with three levels of models

The pragmatic level concentrates on a strategic description, i.e. it is supposed to give a definition of the "why" [26] - a long term intention or a vision of the enterprise under development [4]. A pragmatic description is motivating various enterprise components at the semantic level. Sometimes, desired software components can be easily described before the goals are well understood. Elaboration of the objectives is then done in a backward direction, by asking the reason for existence of introduced components.

Information system semantics can be defined by using the static and dynamic dependencies of various kinds. Most of the semantic diagrams are based on the entity notations that are provided by several links. Links are established to capture semantic detail about various relationships among concepts. The ability to describe information system in a clear and sufficiently rich way is acknowledged as crucial in many areas including software engineering. Typically semantic dependencies are defining semantics in different perspectives such as the "what", "who", "where", "when" and "how" [26]. For instance, in the object oriented approach [13] the "what" perspective is defined by using the class diagram, the "how" perspective can be defined by using the activity diagrams and the "when" perspective in a large part can be described by the state – transition diagrams.

Modelling primitives of the syntactic level depend on the tools that are used in the software development process. Syntactic level of the Lyee methodology is well-understood. It is based on the printout, message, screen and file layouts. These primitives and their dependencies are elements that are used in the definition of a so called process route diagram [14]. It can be used as a definition of a logical structure for a software component source code that is build by the LyeeAll tool. Thus, the syntactic level of the enterprise model in Lyee approach is build on the basis of primitives that are represented in Figure 4.

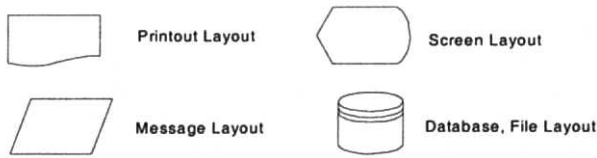


Figure 4. Basic syntactic elements of Lyee approach

It should be noted that syntactic elements are considered as to be the basic building blocks that define the implementation perspective. Therefore, these elements can be defined by using the conventional programming languages or database definition languages. For instance, database relations [6] are typical representatives of the syntactic level.

Humans, technical systems or organisations can be thought as enterprise actors that are dependent on various information, decision or material flows. Flow dependencies among actors are regarded as basic communication patterns, which together with the actor composition or generalisation links describe invisible enterprise architecture (infrastructure). At the pragmatic level, actors are characterised by various goals, problems and opportunities. Actors at the semantic level can be represented by square boxes. They can be viewed as organisational or technical system components. An instance of an organisational component can be a human, group of people, a job-role, a position, an organisation, etc. A technical component can be a machine or a software component. Typical instances of the actors that are relevant for information system development in the Lyee environment are represented by Figure 5.

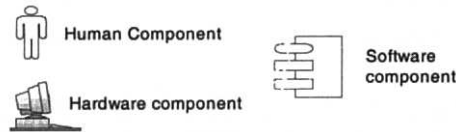


Figure 5. Basic components for definition of the enterprise infrastructure

The presented list of the component classes is not exhaustive. Other types of icons such as fax machines, phones, computer networks, etc., can be introduced on demand. These components in combination with the basic layouts can be used as the main building blocks for representation of the technical deployment architecture, software component infrastructure, organisational dependency structures or general requirements of user interface.

Despite of the fact that the same requirement can be described on the syntactic, semantic and pragmatic level, the granularity of the specification on every level will be different. Information system methodologies recognise that it is not enough to concentrate distinctly on one of the levels. Some information system modelling approaches tend to mix various constructs from the syntactic, semantic and pragmatic levels. This contributes to increased complexity of the information system modelling process. One of the main objectives of the next chapter is to identify and to demonstrate various modelling primitives of the semantic and pragmatic levels.

4. Semantic and Pragmatic Dependencies

Goals of various organisational components stimulate interaction among actors, leading to some further interactions [21]. A typical action workflow loop [1] includes two communication flows sent into opposite directions. An agent is an actor who initiates the

work flow loop to achieve his goal. A recipient of a communication flow can be viewed as an agent in the next communication action. Actor dependencies in two opposite directions imply that certain contractual relationships are established. Various actor communication dependencies are illustrated in Figure 6.

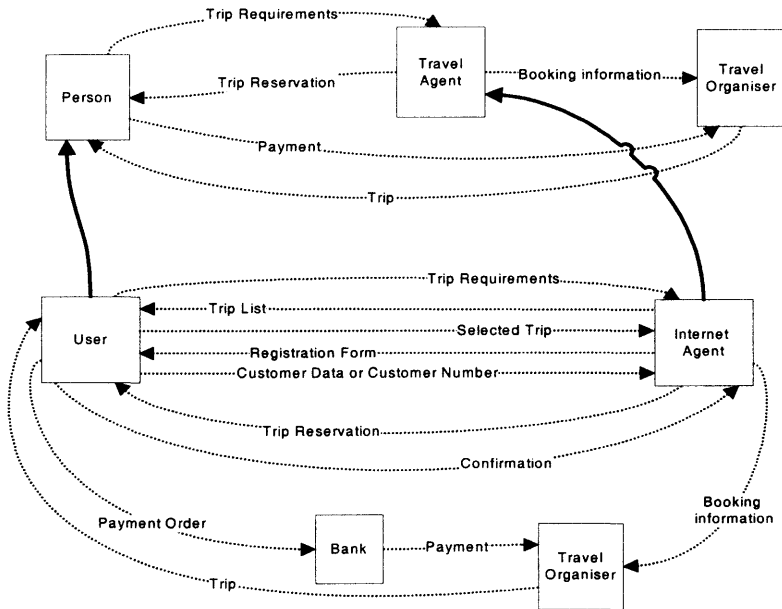


Figure 6. Basic communication flows among actors

The middle part of this diagram illustrates the basic communication flows between User and Internet Agent in the process of making trip reservation using the Travel Agent's web site. This part corresponds to a simplified part of a real process of buying trips on the Internet. The simplifications are not important at this point. A full diagram contains just a bigger number of communication loops including the hotel reservation process for the same trip.

A lower part of the diagram demonstrates that the Bank is involved to handle a payment. A Travel Organiser delivers the actual Trip when the payment for the trip is made. A higher part of the diagram demonstrates the core communication loops in the traditional business process. It is reused from a non-electronic business process of buying trips. The top part can be overridden by the more specific interactions, should a person take a more specific role of a user and an agent would become an Internet Agent. It is not clear at this moment what kind of organisational or technical components can be used to carry out this business process in the real situation.

Any actor can achieve a goal by avoiding a problem. The pragmatic dependencies are used to define various intentions of actors. The goal, opportunity and problem dependencies can be used to refer desirable or not desirable situations. Goals or problems can be decomposed by using a refinement link. It should be noted that the interpretation of some software requirement or situation as a problem, opportunity and goal is relative. The achievement of some goal by one actor can be regarded as a problem for another actor. Negative influence dependency (-) and positive influence dependency (+) are used to indicate influences. Negative influence dependency from A to B indicates that A can be regarded as a problem, because it hinders the achievement of goal B. The positive influence

dependency from A to B would mean that A can be viewed as an opportunity in the achievement of goal B. The basic pragmatic dependencies of the enterprise modelling approach are represented in Figure 7.

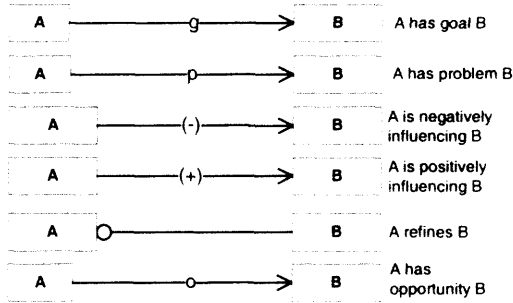


Figure 7. Graphical notation of the pragmatic dependencies

We are not going into details of pragmatic dependencies, because of the space limitations. Some discussion on this issue can be found at [9], [10].

The semantic dependencies in the enterprise modelling are of two kinds: static and dynamic. Descriptions of organisational *activities* as well as *actors* involved in these activities are based on the dynamic dependencies. So, the dynamic part of the enterprise model can be represented by actions using and producing various communication flows and by responsible actors for initiations of those actions. So, the dynamic relations are state dependencies and communication dependencies. The communication dependencies among enterprise actors are relevant for description of the "who" perspective. It is based on the communication action tradition in information system development that is stemming from Scandinavian approaches. Actor dependencies are described as action and communication links in the enterprise model. Graphical notations of dynamic dependencies are represented in Figure 8.

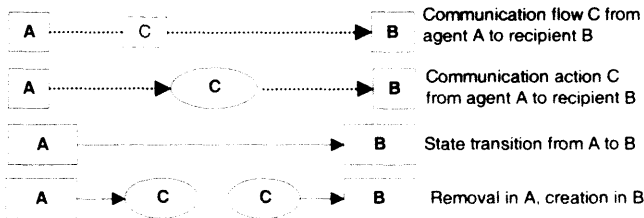


Figure 8. Graphical notation of the dynamic dependencies

State dependencies define semantic relationships between states of actions. They are normally considered as the "how" perspective. Both state transition and communication dependencies describe a very important part of knowledge about business processes. Unfortunately, many communication approaches often neglect some behavioural aspects of the state transition and vice versa, many software engineering approaches disregard the dependencies of communication.

The static concept dependencies are used for specification of the attributes for various states of processes. They define the "what" perspective. A static part of the enterprise model can be defined by using dependencies of two kinds: intensional and extensional. The extensional dependencies are defining the relations between concepts and their instances.

The intensional dependencies are stemming from various semantic models that are introduced in the area of information system analysis and design. It means that all kinds of conventional static relations can be defined between concepts such as classes, actors, states or flows. Semantics of static dependencies can be defined by the cardinality constraints [13]. The similarities can be shared between concepts by extracting and attaching them to a more general concept. In this way, all kinds of static and dynamic dependency links can be inherited by several concepts. Composition or aggregation dependency is useful for formation of a new concept as a whole from other concepts that might be viewed as parts. Graphical notations of static dependencies are represented in Figure 9 (see examples in the next figure).

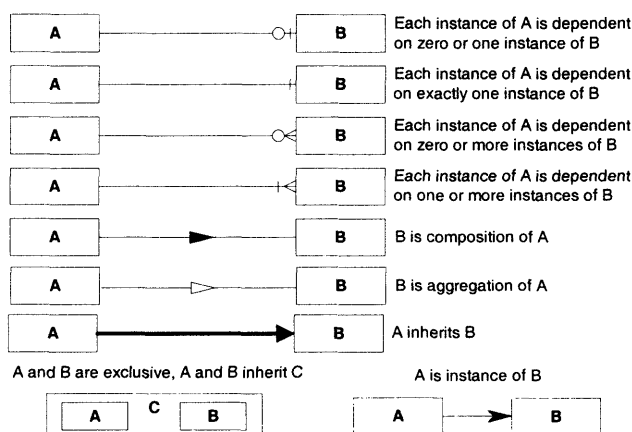


Figure 9. Graphical notation of the static dependencies

Most methodologies of information system engineering do not allow for a concept to play different semantic roles. The problem of strict classification of semantic roles is one of the main reasons of severe difficulties in the area of view integration [2]. Strict classification of concepts is a cause of structurally different, but semantically equivalent representation. Such differences can be justifiable in the information system design phase, because they show some important implementation details. Semantically equivalent, but structurally different representations violate the independence principle - a main principle of conceptualisation [8]. Moreover, such discrepancies create headache for analysts during the process of information system requirement engineering.

The stance of various designers on how accurate the semantic roles are may differ significantly. Despite of this fact, the conventional methods of system analysis and design enforce early decisions on various interpretations of concepts. It may create situation of misunderstanding among various designers [11]. In the enterprise modelling approach [9], [10], the interpretation of semantic roles is flexible. Whether a concept is regarded as an instance, a class, a communication flow, a state, a problem, a goal or an actor depends upon types of the semantic links these concepts are related.

Dynamic semantic dependencies are used to define relations between different actors, their actions and communication flows. If concept A is connected to B by a communication dependency, then A is an agent and B is a recipient. Depending on whether there is or there is no physical flow component in the action, the communication flows can be information, material or decision. The physical flows represent instances of objects to be transferred from agent to recipient. Two extended communication loops between User and Internet Agent are represented in Figure 10.

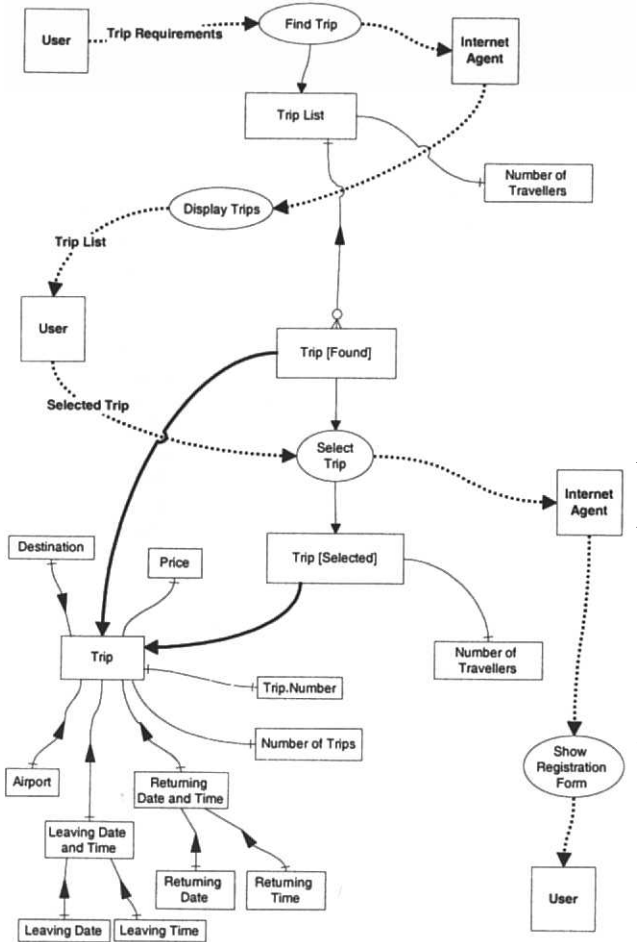


Figure 10. Semantic Dependencies of the reservation process

It should be noted that the presented diagram is incomplete, but it is consistent with the basic communication diagram that was previously defined (see the more abstract diagram).

Any communication loop is able to change the static associations between instances. State changes are important to both actors. Without the ability to represent noteworthy state changes [7], we would have difficulties to understand the rational and effect of every communication loop. Actions express the permissible ways in which state changes may occur. These changes are specified by using the integrated dependency communication and state transition [10].

5. LYEE Layouts as A Design Consequence of the Enterprise Model

Cohesion of action in terms of state transition and communication dependency results into the abstraction that may reveal incompleteness or redundancy of the state attributes. Cohesion analysis is questioning the integrity between computerised information system state change and communication flow internal structure. Any parameter of a communication flow is supposed to be used for some purpose. It might be the creation of an object or

the deletion of an object in some state. A removal communication action for a specific state must destroy all associations that are relevant for this state. A creation must establish all relevant associations, otherwise the action is incoherent. It should be noted that some actions can create and destroy associations at the same time. Such internal changes can be followed according to the static dependencies defined for various states.

Communication flow parameters can be consumed or emitted by a predefined action. A consumption action is supposed to compare or to allocate the parameter values into the matching state attributes. Every consumed parameter has to be defined in a postate of action by using the same or a different name. If this name is different, then an explicit computation rule must be introduced. Usage of any attribute should be questionable with respect to an overall structure incoming flow. If an attribute is irrelevant for the action, then it should be removed. Termination process can be represented by an action without a resulting state. In this case, no commitments of actors should be pending in connection to this state. A properly designed termination should emit values of the attributes by a communication flow that can be viewed as a message to the initiator of the previous action.

The communication flow structures with the action names and the generalised state structures define requirements for specification of defineds in Lyee methodology. For instance, the dependent concepts of Trip (see previous figure) constitute the foundation of a file layout that can be represented by the following relation:

TRIP (Trip_Number, Destination, Airport, Leaving_Date, Leaving_Time, Returning_Date, Returning_Time, Number_of_Trips, Price).

When *Find Trip* action is receiving parameters from a *User*, a *Trip List* object is created, which consists of 0 or many *Found Trips*. Any *Found Trip* should satisfy the specific computational rules. If these rules are not defined, then the action is supposed to allocate the received parameter values into a space that is defined by the attribute structure. The next action would create a *Trip List* (note: definition of *Display Trips* action from the point of view of enterprise modelling is incomplete). *Trip Requirements* communication flow structure is presented by Figure 11.

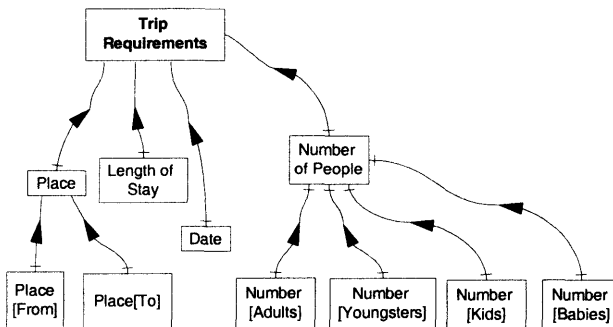


Figure 11. Trip Requirement communication flow structure

This structure with all possible action names that could affect the illustrated communication flow parameters, define the static requirements of the first screen layout. For the reason of space limitations, the contingent actions are not presented in this paper.

The first screen layout¹ at the syntactic level will be identified by Screen 1. It is represented in Figure 12.

Figure 12. Screen layout (Screen 1) of Trip Requirements

Various constraints that need to be implemented by software component in Lyee approach are entitled to as Logical Formulas. At the implementation independent level of enterprise modelling approach, those constraints are defined as the computation rules. The computation rules, which were identified for a Find Trip action, are as follows:

Trip[Found]. Leaving Date \rightarrow Find Trip (Trip Requirements.Date), Trip[Found]. Returning Date \leq Find Trip (Trip Requirements.Date + Trip Requirements.Length of Stay), Trip[Found]. Destination \rightarrow Find Trip (Trip Requirements.Place[To]), Trip[Found]. Airport \rightarrow Find Trip (Trip Requirements.Place[From]), Trip [Found]. Number of Trips \geq Find Trip (Trip List.Number_of_Travellers),
 Trip List.Number_of_Travelers = Find Trip (Trip_Requirements . (Number[Adults] + Number[Youngsters] + Number[Kids] + Number[Babies]).
 Trip List = Display Trips (Trip_Requirements.{Trip[Found]}).

Here: \rightarrow is inheritance dependency (see dependencies). We are not going any further into the semantic issues of computational rule definition for the reason of the space limitations.

The enterprise modelling dependencies can be viewed as modelling technique to refine and analyse the syntactic relationships. Those are constrained by the dependencies at the semantic level. We have presented a small illustration of how the TRIP file layout and the Screen1 layout was defined. These layouts are defining the operating infrastructure of the Internet Agent software component. A relevant part of the Internet Agent enterprise model at the syntactic level is represented in Figure 13.

¹ We would like to thank Lars Jakobsson for implementation of the Internet Agent software component that was defined by using the enterprise modelling and the Lyee approach.

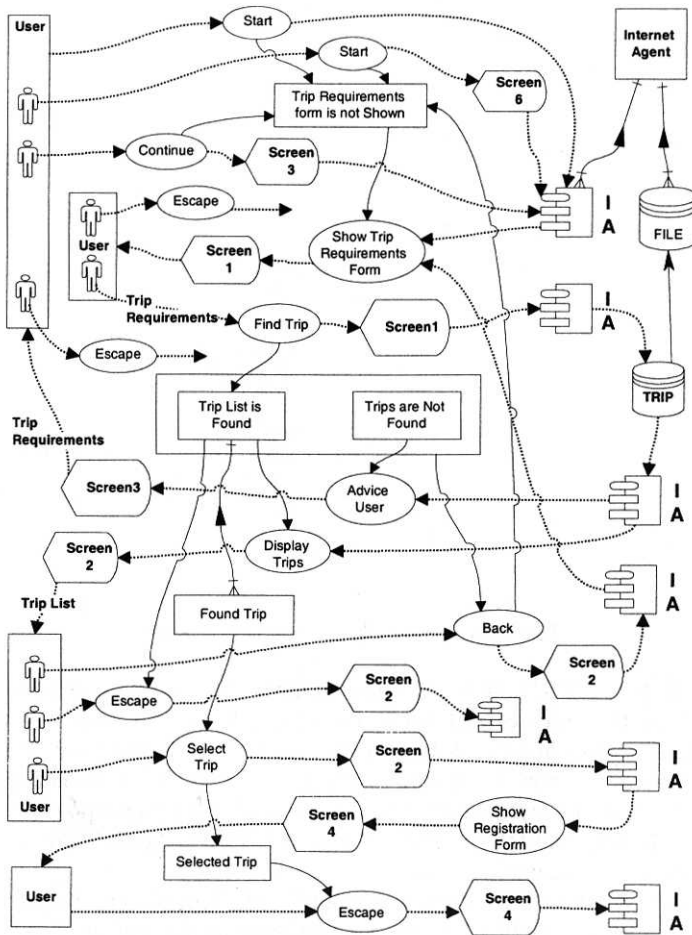


Figure 13. A syntactic diagram of the IA (Internet Agent) software component

This diagram illustrates how the semantic part of the enterprise model is implemented. It should be noted that various previously defined dependencies have to be taken into account by a more specific level. It means they have to be consistent on various levels of abstraction. A special inference rules can be defined in order to validate the semantic consistency. In this case, the enterprise modelling framework can be used as a uniform basis of reasoning about unambiguity [9], coherence and completeness of software processes that are defined on the neighbouring levels of abstraction.

The logical diagrams can be used to define a Process Route Diagrams (PRD) that is a basic syntactic level representation of the Lyee methodology (we are not going into details any further). A PRD specifies the navigational links among various building blocks, which compose graphical description of a software component. The edges of PRD define the sequence or branching of scenario function triples. Understanding and designing of a PRD from the natural language description is a very complicated task. A good illustration of how the PRD is developed can be found in the case study [17].

A PRD represents a basic behaviour of the software component. This behaviour is rooted in a structurally richer specification that is shown in the previous diagram. A corresponding PRD is represented in Figure 14.

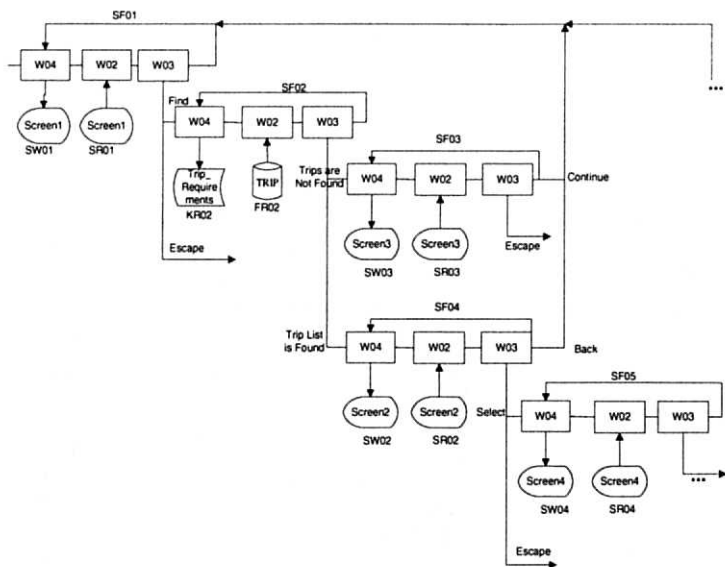


Figure 14. A part of the Process Route diagram

The represented part of PRD shows the files and main interactions in terms of screens that are designed for the graphical interface. The diagram contains five scenario function triples. Each of them consists of three pallets. WO4 is dealing with the display of information flow to a user or storing information to a file or a database. WO2 is responsible for accessing information and execution of the computation rules. The third pallet WO3 is dealing with the recovery of input information flows and navigating to the successive triples of the scenario function.

The pallets are composed of words and grouped into logical units on the basis of the enterprise model at the semantic level. Specification of the logical units serves as a software component logical definition in the LyeeAll tool. Logical units define the behavior of the screen, file, message or printout layouts. The logical units are identified according to the following rules:

- SR – reading data parameter structure from a screen layout,
- SW – writing data parameter structure to a screen layout,
- FR – file retrieval parameter structure,
- FW – data modification parameter structure,
- KR - database query parameter structure (select statement),
- KW – database query modification parameters (update or delete statement).

From the software component builder’s perspective, the definition of the logical part of it is not sufficient. The builders need to understand the technical system architecture and organizational infrastructure, where application is going to be installed. Therefore, the enterprise modelling approach focuses not just on the on the pragmatics, semantics and how the logical structure of software components, but also how these components are deployed.

6. Concluding Remarks

The goal of this paper was to demonstrate the possibility of bridging the Lyee methodology to the enterprise modelling approach. One of the main objectives of

extending the Lyee methodology is adaptation of the graphical models that might be helpful in solving information system development and new software development problems. The enterprise models can be used for change analysis in a systematic way on a basis of the graphical representations that are defined for both traditional and electronic business.

We would like to emphasise that the described ideas should be considered as initial findings in the area that lies in the intersection of software requirements engineering by using the Lyee methodology and enterprise modelling approach. Most software methodologies are heavily centred on system development issues at the implementation level. Thus, such methodologies are restrictive in a way that a supporting technical system specification can not be motivated or justified in the context of organizational process models. The Lyee methodology is not an exception. The process route diagram that is the main logical structure used in the input of LyeeAll is not easy to use for reasoning, when changes need to be introduced. Enterprise models provide a basis for the gradual understanding of why and how various layout, computation rule and process routes come about.

The major focus of this paper is on how the syntactic elements of Lyee methodology can be motivated by using the enterprise modelling approach. It demonstrates a way in which various pragmatic, semantic and logical software requirements can be defined and integrated. Sometimes, global discontinuities between organisational and software processes, redundancies [26] and inconsistencies being introduced in the enterprise as a whole may cause more problems than local benefits. Every enterprise component must be structurally and behaviourally consistent with respect to a whole. We believe a new systematic way of dealing with various software requirements is contributing to the development of the extended approach for software engineering.

We expect that the enterprise models can be used as a core method to analyse the rationale of the new organisational solution prior a new supporting IT system is introduced. It should help managers and IT experts to define, visualise and to assess various organisational changes by using a fully graphical approach to business process reengineering. This would in turn facilitate justification and motivation of software components that are used to support work of organisational actors involved in various business processes. Enterprise models can be considered as a corporate resource in diagnosing potential problems, these models are crucial to enable reasoning about business process integrity and the purposeful implications of an organisational change.

References

- [1] Action Technologies. (1993) *Action Workflow Analysis Users Guide*, Action Technologies.
- [2] Batini, C., Lenzerini, M. & Navathe, B. L. (1986) "A Comparative Analysis of Methodologies for Data Base Schema Integration", *ACM Computing Surveys*, Vol. 18, No.4, pp. 323 - 363.
- [3] Booch, G., Rumbaugh, J. & Jacobsson, I. (1999) *The Unified Modelling Language User Guide*, Addison Wesley Longman, Inc., Massachusetts.
- [4] Bubenko, J. A. (1993) "Extending the Scope of Information Modelling", *Fourth International Workshop on the Deductive Approach to Information Systems and Databases*, Polytechnical University of Catalonia, 73-97.
- [5] Checkland, P. B. (1981) *Systems Thinking, System Practice*, Wiley, Chichester.
- [6] Elmasri, R. (1994) *Fundamentals of Database Systems*, Benjamin-Cummings, 1994.
- [7] Harel, D. (1987) "Statecharts: A Visual Formalism for complex Systems", *Science of Computer Programming* 8, Elsevier Science Publishers, North-Holland, 231-274.

- [8] Griethuisen, J. J. (1982) *Concepts and Terminology for the Conceptual Schema and Information Base*. Report ISO TC97/SC5/WG5, No 695.
- [9] Gustas, R. (1998) "Integrated Approach for Modelling of Semantic and Pragmatic Dependencies of Information Systems", *Conceptual Modelling - ER'98*, Springer, pp. 121-134.
- [10] Gustas, R. (2000) "Integrated Approach for Information System Analysis at the Enterprise Level". *Enterprise Information Systems*, Kluwer Academic Publishers, pp. 81-88.
- [11] Gustiene, P. & Gustas, R. (2002) "On a Problem of Ambiguity and Semantic Role Relativity in Conceptual Modelling", Proceedings of International conference on Advances in Infrastructure for e-Business, e-Education, e-Science, and e-Medicine on the Internet, ISBN 88-85280-62-5, L'Aquila, Italy.
- [12] Jackson, M. A. (1983) *System Development*, Prentice Hall, Englewood Cliffs, N.J.
- [13] Martin, J. & Odell, J. J. (1998) *Object-Oriented Methods: A Foundation (UML edition)*, Prentice-Hall, Englewood Cliffs, New Jersey.
- [14] Negoro, F. (2001) "Methodology to Define Software in a Deterministic Manner", *Proceedings of ICII2001*, Beijing, China.
- [15] Roland, C., Prakash, N. & Benjamin A. (1999) A Multi – Model View of Process Modelling, *The Requirements Engineering Journal* (to be published).
- [16] Roland, C. & Prakash, N. (2001) From Conceptual Modelling to Requirements Engineering, *Annals of Software Engineering* (to be published).
- [17] Roland, C., Salinesi, C., Ayed, M., B., Nurcan, S., Kla, R., Nalazawa, A., Nakamura, K. & Takada, H. (2001) *Development Using Lyee: A Case Study with LYEEALL*, Technical report, University of Paris and The Institute of Computer Based Software Methodology and Technology.
- [18] Snoeck, M., Dedene, G., Verhelst, M. & Depuydt, A. M. (1999) *Object-Oriented Enterprise Modelling with MERODE*, Leuven University Press.
- [19] Storey, V. C. (1993) "Understanding Semantic Relationships", *VLDB Journal*, F Marianski (ed.), Vol.2, pp.455-487.
- [20] Vernadat, F. B. (1996) *Enterprise Modelling and Integration: Principles and Applications*, Chapman & Hall, London.
- [21] Warboys, B., Kawalek, P., Robertson, I. & Greenwood, M. (1999) *Business Information Systems: A Process Approach*, McGraw-Hill Co., London.
- [22] Winter, M. C., Brown, D. H. & Checkland P. B. (1995) "A Role of Soft Systems Methodology in Information System Development", *European Journal of Information Systems*, No. 4, 130-142.
- [23] Yourdon, E. (1989) *Modern Structured Analysis*, Prentice-Hall, Englewood Cliffs, N.J.
- [24] Yu, E. & Mylopoulos, J. (1994) "From E-R to 'A-R' - Modelling Strategic Actor Relationships for Business Process Reengineering", *13th International Conference on the Entity - Relationship Approach*, Manchester, U.K.
- [25] Zachman, J. A. (1987) "A Framework for Information Systems Architecture", *IBM Systems Journal*, vol. 26, No. 3.
- [26] Zachman, J. A. (1996) "Enterprise Architecture: The Issue of the Century", *Database Programming and Design Magazine*.

Extending Process Route Diagrams for Use with Software Components

Lars JAKOBSSON

Karlstad University, Universitetsgatan 1, 651 88 Karlstad, Sweden

Abstract -- This paper describes the differences in software development analysis using the Lyee methodology and traditional development methods and programming languages in conjunction with software components. Emphasis will be on extending Process Route Diagrams to better be able to understand and describe software under development using Lyee and a component-based approach, with influences from Enterprise Modelling. The paper proposes possible extensions to the Lyee Process Route Diagrams to better facilitate modelling of software using software components.

1. Introduction

When developing software for any kind of platform, we need to use a programming language and a structure suitable for the specific platform. We also need to analyse the requirements of the system and the current situation in order to be able to design a software product that complies with the customer demands and the demands from the surrounding systems. There are several methods for analysing requirements today, and all of them have advantages as well as disadvantages. In the Lyee methodology, requirements analysis is done by designing Screen Layouts with the user, placing the Screen Layouts in sequence and based on the Screen Sequence produce a Process Route Diagram describing the system being developed.

Another way of analysing requirements and structures in the system to be developed is by using the Enterprise Modelling Approach by R. Gustas [1]. Regardless of which approach is used it is important to analyse static and dynamic parts of the system as well as semantics, completeness and consistency. To be able to analyse the system as a whole, and to address completeness and consistency issues, there is a need to be able to model the system as a whole. This can easily be achieved by using the Enterprise Modelling Approach [1].

This paper is based on a simple E-commerce system developed for testing the Lyee methodology in a software component environment. Based on the sample program, consisting of a main application, one database and a software component, this paper will focus on extending the Process Route Diagram with influences from Enterprise Modelling approach in the context of using a component-based approach when developing software.

When developing software using software components it is imperative to be able to understand how the software components intended for use within the development are supposed to be used, as well as understanding the overall system. A modelling technique well suited for this is the Enterprise Modelling Approach [1].

2. Software Component Definition

One of the more recent techniques in the software development industry is component-based software development. This means that information systems are created through assembling more or less standardised software components into a unique software solution. The term software component isn't easy to define, it does not have a clear-cut definition in the software development community, but the meaning fluctuates. There is no clear-cut definition on what a software component actually is, but there is some consensus on what a software component should be like.

"A component is a unit of software of precise purpose sold to the application developer community...with the primary benefit of eliminating a majority of the programming the buyer must perform to develop one or more function points..." [2].

"A component is a reusable piece of software in binary form that can be plugged into other components from other vendors with relatively little effort" [3].

"A software component...:

- is independent and reusable,
- offers explicitly specified services through an explicitly specified interface,
- can affect/be affected by other software components,
- should have one documented specification (the software component described in a high level of abstraction),
- can have several independent implementation, i.e. one component can be implemented in several different programming languages, and
- can have several executable (binary) shapes, i.e. one component can be executed in different software environments." [4].

Since the definition provided by Christiansson [4] cover the other definitions by Steel [2] and Microsoft [3], but is more precise, the definition by Christiansson will be used in this paper.

3. Component specification

The software component, developed for the purpose of analysing Lyee's ability to cope with software components, is a simple data storage and retrieving facility, for accessing the database containing the Trips. The component can take input from any other component or program and store data and values needed to fulfil the requirements of the final output to the database. The component also has a specified interface to which all communication with the component must be directed. The software component, named LMJReservationComponent, is implemented as a .DLL-file.

In order for the comparison between LyeeAll2 and "normal" development to be as accurate as possible, the application for testing is developed using the Microsoft Visual Basic 6.0 SP 5 environment. The storage component is implemented as a class in Visual Basic, and deployed as a stand-alone software component DLL.

Error checking (normally found in applications) is deliberately not addressed, because it is not necessary to address these issues in this paper. Some simplifications in functionality are made for same reason. Because of space limitations these issues will not be discussed further. In this paper, only a smaller part of the system is described and discussed as this is sufficient for the purpose of the paper.

Table 1 and Figure 1 describes the interface to the component in terms of public methods in the class contained in the software component. The table is formatted to be similar to tables used in the Lyee methodology used to specify defineds, and the figure represents the part of the component used in this paper in UML-style, according to Bennett [5].

Table 1. Public methods of the LMJReservationComponent

Names	Return Values	Type	Parameters	Type
FindTrip	None	None	From, To, Leaving_Date, Returning_Date, Number_Of_People	String
DisplayTrips	Trip_Number, Destination, Airport, Leaving_Date, Leaving_Time, Returning_Date, Returning_Time, Number_Of_Trips, Price	String	None	None

Noticeable for the methods, FindTrip as well as DisplayTrips, the parameter in the case for FindTrip, and the return value for DisplayTrips, is that the message structure is in the form of an array of strings. This is possible to understand from the table by the fact that there is only one type for both FindTrip and DisplayTrips. What is not shown in the table is that there is a specific interface to the component and it is not possible to tell which attributes are included in the class or the scope of the methods. The cardinality between the interface and the class itself is not apparent either.

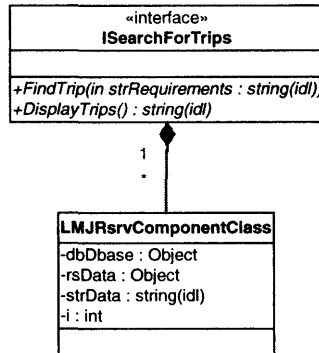


Figure 1. UML Class diagram describing the LMJReservationComponent

The UML class diagram describes both the interface as well as the class of the component, and the relation between these. The interface of the component is shown with the public (a + sign in front of the method name indicates “public” in UML) interface methods used to access the component class. Further, the figure indicates that the interface can have several instances of the class LMJRsrvComponentClass. The diagram in Figure 1 indicates that the class has four attribute members. All attributes shown in the diagram are private (indicated by the – sign). The (idl) following the data types in the diagram indicates that an array of the data type is used. For example the “+FindTrip(in strRequirements : string(idl))” indicates that the FindTrip method takes an array of strings as an argument to the function call, and that no output parameters are used in this method. DisplayTrips has a return value

of an array of strings. Missing in the diagram is the possibility to indicate in which order the items in the array should be organised.

Only the methods and attributes used in relation to this paper are included in the diagram, this goes for the table as well. There are several methods and attributes contained in the component, which do not apply to this paper, and they are therefore omitted from the diagram and the table.

4. Enterprise Modelling Approach

In the Enterprise Modelling Approach the goal is to describe the whole of the system in one diagram, enabling for consistency and completeness. The same notation can be used for describing the system regardless of at which granularity level we want to make the descriptions. In this paper, only a brief legend of the Enterprise Model will be presented. The symbols used are represented in Figure 2.

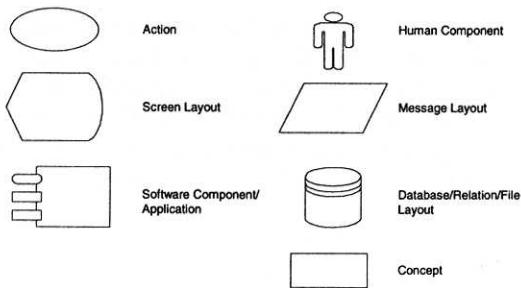


Figure 2. Symbols used in the Enterprise Modelling approach

Any actor in the model can invoke an action. Actors are in this case Human Component, Software Component and Application. There is a distinction between Software Component and Application, where the software component is more specific than Application. The Application does not comply with all criteria found in the beginning of this paper discussing the definition of Software Component. Applicable for both Application and Software Component is that they reside somewhere in the software realm of the system. Screen Layouts are used to indicate where the Human Component is supposed to enter information to and receive information from the system. The Human Component is an actual human being, residing in the realm outside the software system, but within the system as a whole. The Message Layout is used to indicate that an action directed to a component is containing a structured message of some kind. The Database/Relation/File Layout indicates that some sort of structured storage facility is involved, usually situated in some sort of magnetic media, but that is not necessarily the case. The Concept icon is used in many different ways in the Enterprise Modelling Approach, in this paper it is used as attribute, pre/post-condition and conditional constraint.

4.1 Enterprise Model

The Enterprise modelled part of the system in this paper handles the scenario where the user uses the system to search for a trip in the systems database. The communication to the

software component responsible for searching the database is handled by the main application. The user interacts only with the main application, and this is the case for the software component as well. To the user the software component should not be “visible” during run-time, as all communication to it is hidden by the main application. However, for the developer and the user to be able to properly discuss the system, the component should be visible in the model describing the system.

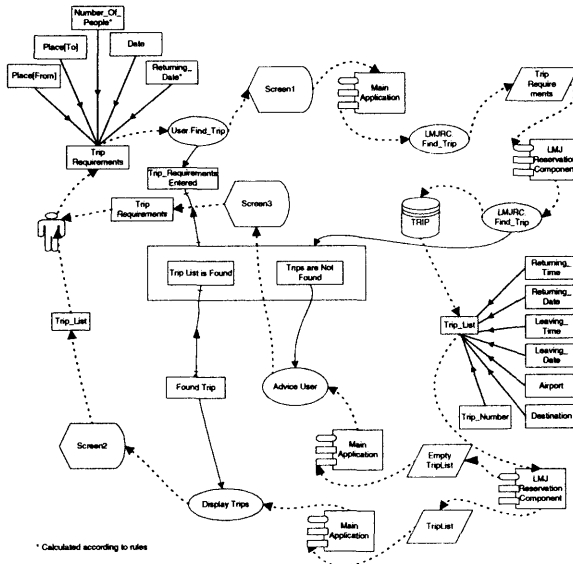


Figure 3. Part of the system described using Enterprise Modelling

In Figure 3, there are two essential message flows between the main application and the software component: *Trip_Requirements* and *Trip_List*. *Trip_Requirements* are used as input to the software component to search for matching trips in the database. *Trip_List* is used to return the search result to the main application for display to the user. If the *Trip_List* is empty, this is used as an indication that no matching trips were found. The items marked with an asterisk are actually calculated from other data according to the rules:

$$\text{Number_Of_People} = \text{Number}[\text{Adults}] + \text{Number}[\text{Youngsters}] + \text{Number}[\text{Kids}] + \text{Number}[\text{Babies}]$$

$\text{Returning_Date} = \text{Date} + \text{Length_Of_Stay}$

The items *Number[Adults]*, *Number[Youngsters]*, *Number[Kids]*, *Number[Babies]*, *Date* and *Length_Of_Stay* are taken from the Screen1 screen layout shown in Figure 3. For a description of the complete system it is recommended to use Enterprise modelling according to Gustas, Gustiene [6].

4.2 Mapping the Enterprise Model to component logical structure

The user sends the *Trip Requirements* to the main application by clicking the “Find_Trip” button, after having entered the trip requirements in Screen1. The main application invokes the method *Find_Trip* on the LMJ Reservation Component (LMJRC) by sending the

The data used by the application developed is stored in some kind of file, most commonly databases are used for this.

5.1 Process Route Diagram – Current State

Based on the sequenced screen layouts, a schema describing the processing routes can be developed. This schema is called Process Route Diagram (PRD), and is used as the main input to the LyeeALL2-tool. There are five basic concepts used in a PRD: Scenario Functions, Screen Layouts, File Layouts, Keys and Links. The Scenario Function is composed of three “Pallets”: W04, W02 and W03, where W04 is used for output, W02 is used for input, and W03 is used for processing, Negoro [7]. The system described earlier using enterprise modelling is described using PRD in Figure 5.

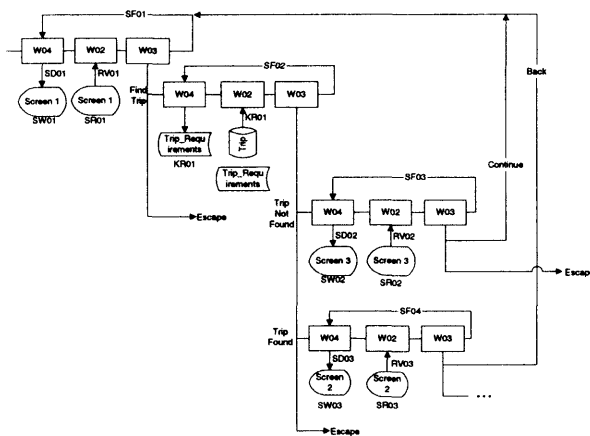


Figure 5. Process Route Diagram describing part of the system

The PRD shown in Figure 5 depicts the same part of the system as the Enterprise model in Figure 4 does. However it is not as detailed as the Enterprise model, and it is incomplete, because the impression is that all functionality is included into one application. One of the items missing is the software component. The PRD does not take into account whether software components are used or not, which means that fruitful discussions on the system are difficult to maintain, because you can not see what parts the system is built from or how interaction between these parts should be maintained. The Process Route Diagram only illustrates the route of processing through the overall system without taking into account the parts of the system (parts here meaning software components used to compose the system). The PRD is not intended to depict the system as a whole, but is centred on the screen layouts, which leads to the problem of incompleteness.

5.2 The Process Route Diagram Explained

Firstly, Screen1 is shown to the user, who enters information about his/her Trip Requirements. When the user clicks “Find Trip” (a button found on Screen1, for which the design is not shown in this paper), the data entered by the user on Screen1 is read by W02.

The Trip database relation is searched for data matching the key "Trip_Requirements" (KR01). In fact, the KR01-key is composed from the data read from Screen1, but that is not shown in the PRD. If one or more matching trips (represented in the Enterprise model as Trip_List) are found in the Trip database relation, Screen2 is shown to the user. In Screen2 the user has an opportunity to choose one trip from the Trip List. If no matching trips are found, Screen3 is displayed to the user, where he/she has the opportunity to review the searched trip data.

6. Process Route Diagram Limitations

Process Route Diagrams in the current state does not allow for modelling of software components, as there is no symbol representing software components. There is also no possibility of describing the interface to the software components used. Furthermore the messages needed to communicate with software components can not be described in the current Process Route Diagrams. There is no clear connection between the read data from screens and the keys used for searching files in Process Route Diagrams in the current version. The Process Route Diagrams may be redundant when two or more possible routes can be derived from one screen, where part of the following route is the same regardless of what the route from the screen is.

There is of course a possibility to use software components in a system developed using the LyeeAll-tool, as it is possible to manually achieve communication through the Action Vectors, but it is not possible to describe this with the current state Process Route Diagrams. There is of course also the possibility to achieve component communication by modifying the source code generated by the LyeeAll-tool, but this is against Lyee methodology and definitely not recommended, and should therefore not even be considered.

The fact that there is no support for message layouts and software components (or other external communication for that matter) means that it is very difficult to discuss semantics and consistency on the basis of Process Route Diagrams. It is difficult to discuss the system as a whole based only on the Process Route Diagrams.

Ackoff has identified that one of the causes for misunderstanding between developer and user is that it is very difficult for the developer to obtain a complete understanding of how an overall system is working [8]. Models describing the system as a whole can aid in avoiding this misunderstanding as models tend to be less ambiguous than natural language [6]. A common problem according to Bennett et al [5] is that users/clients change the requirements frequently, which in turn leads to problems in change management during the development process. A common statement from developers is "We built what they *said* they wanted" [5]. To remedy some of the problems generated by this statement, graphical notation can be used to model the system. This would give the user/client a clearer view of the system discussed and hopefully helps avoiding too many changes of the requirements.

6.1 Extensions to the PRD

To remedy the limitations of the Process Route Diagrams of current state, and to enable possibility to check integrity, consistency and completeness, there is a need of extending the Process Route Diagram. The extension suggested in this paper is mainly intended for use with software components, but could possibly be applied more generally, for instance

for more descriptive communication to a DBMS. Firstly one symbol, representing message layout structure would be added to support usage and description of messages between application and software component. The message symbol will be needed to describe message structure and communication between the application developed by using the LyeeAll-tool and the software components. The symbol can be equivalent in design as the symbol of the message layout shown in Figure 2.

Introducing only this new symbol of message layout will not imply any radical changes in the Process Route Diagrams, but will rather make them more complete as to describing the system in conjunction with software components. A proposal to how the extended Process Route Diagram could look like for the system described in this paper is shown in Figure 6.

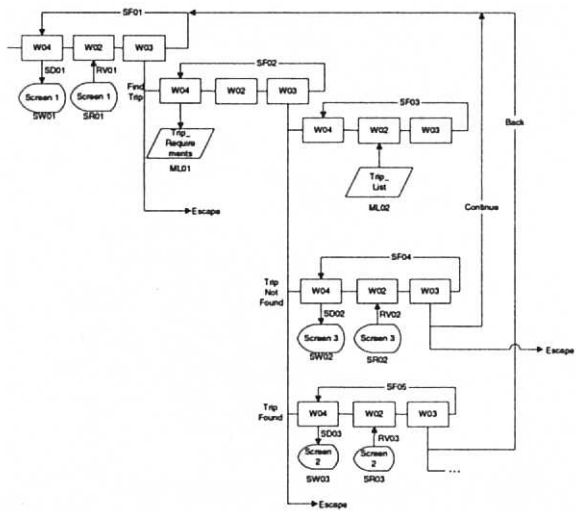


Figure 6. PRD with extensions

The extended PRD shown here has the advantage over the traditional one as the message layout is included in this version. In SF02, the communication to the software component is initiated with the message ML01. The message from the software component to the main application is represented in SF03 by the message layout ML02. Also to be noted is that the message layouts (ML01, ML02) have names matching the Enterprise model shown in Figure 3. The message structure itself is not included in the diagram. The structure and contents of the messages can easily be described as shown in the next section of this paper.

6.2 Describing Message Layouts

The message layouts could be described using any format e.g. XML SOAP, but in this paper a table will be used, similar to table of defineds in the Lyee methodology. In the example described in this paper Table 2 corresponds to the message layouts shown in the Enterprise model.

Table 2. Message Layouts

Message	ID	Fields	Type
Trip_Requirements	ML01	Place[From], Place[To], Number_Of_People, Date, Returning_Date,	String

Trip_List	ML02	Trip_Number, Destination, Airport, Leaving_Date, Leaving_Time, Returning_Date, Returning_Time	String
-----------	------	---	--------

This table represents the structure of the messages ML01, ML02, where the message actually is an array of strings for both messages. The column “Message” contains the name of the message used in the Enterprise model, and the “ID” column contains the ID used in the Process Route Diagram for easy mapping between Enterprise model and PRD.

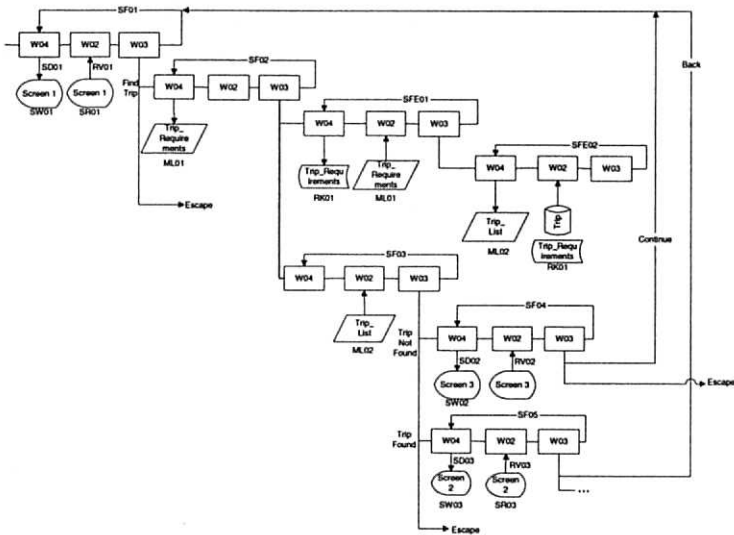


Figure 7. PRD with software component included

The diagram shown in Figure 7 will not be suitable for implementation using the LyeAll-tool of the current version, as the external scenario functions will probably be confusing for the tool, although it is more descriptive and a closer match to the more complete Enterprise Model. Instead the simpler diagram in Figure 6 should be used with the LyeAll-tool of today.

Noticeable is that the software component is represented as external scenario functions, see SFE01, SFE02 in the figure. The external Scenario Functions are in this case Scenario Functions residing within the software component and they are invoked by explicit calls to methods. Ideally, the symbol for the external data processing should be replaced by the component icon, shown in Figure 2, for a better overview and understanding. Though the whole system should not be represented in the PRD [7], the software component used is visible in this suggested extension to the PRD. There is of course also the opportunity to make a PRD like the one in Figure 6 to describe the structure of the part of the system developed by using the LyeAll-tool, and a separate PRD for the software component. However, recent studies indicate that it is more confusing for the developer as well as for the user/client to switch between different diagrams than keeping everything in a larger, well-organised model Turetken, Schuff, [9].

Thus, for discussions on the system the diagram in Figure 7 is more suitable than the one in Figure 6. Notice that the simpler extension (Figure 6) to the Process Route Diagram only include the message layout used for communication between the LyeAll-tool and the

software component. The software component itself and the database used by the component are not included. This of course means that the description of the system is not complete, and Enterprise modelling could be used to provide a more complete description of the system if the simpler PRD is used.

7. Concluding Remarks

To better facilitate for software component use with the Lyee methodology, some extensions to the PRD are suggested in this paper. The aim of the extensions is to graphically incorporate software components into Process Route Diagrams to ensure that mapping between Enterprise models and PRD is possible. Mapping more precisely between Enterprise modelling and Process Route Diagrams enables consistency, ambiguity and completeness control.

Two versions of extensions are suggested in this paper. One is more complete than the other, but the less complete suggestion is more compatible to the existing version of the LyeeAll-tool. The more complete suggestion, where both message layouts and software components are included is better suited for analysis and discussions on the system and is more resistant to incompleteness issues.

As a first step the extensions shown in Figure 6, with only the message layout added, should be used in combination with a separate Process Route Diagram describing the software component when designing the system. The next step should be to perform an analysis on whether the LyeeAll-tool can be used in conjunction with the extensions suggested in Figure 7, and how the external Scenario Functions could be specified.

The question on how much to extend the Process Route Diagrams is a matter of delicacy. The PRD in the state that it is in today is very easy to understand for a novice with a little help from an experienced developer familiar to the Lyee Methodology. This is strength in the PRD, as very large diagrams tend to be difficult to understand, and larger diagrams are time consuming to create, thus more costly. At the same time the PRD is not suitable for deeper analysis of the systems integrity because they are not very detailed in describing the system. Enterprise models are more suited for this. Mapping between Enterprise modelling and PRD is one way of dealing with the problematic of analysing integrity of a software system being developed according to the Lyee methodology. With the suggested extensions to the PRD it is possible to map between Enterprise Modelling and PRD without losing too much of the information contained within the Enterprise Model in a development process where software components are included.

References

- [1] Gustas, R. (2000) "Integrated Approach for Information System Analysis at the Enterprise Level", Enterprise Information Systems, Kluwer Academic Publishers, pp. 81-88.
- [2] Steel J. (1996). *Component Technology Part I An IDC White Paper*. International Data Corporation, UK, London.
- [3] Microsoft. (1996) *The Microsoft Object Technology Strategy: Component Software*. www.microsoft.com.

- [4] Christiansson B. (2001) *Component-based systems development – a shortcut or the longest way around?*, in Nilsson, A. G. & Pettersson, J. S. (eds.) *On Methods for Systems Development in Professional Organisations – The Karlstad University Approach to Information Systems and its Role in Society*, Studentlitteratur, Lund, .
- [5] Bennett, S., McRobb, S., Farmer, R. (2002) *Object-Oriented Systems Analysis And Design Using UML*, 2ed, McGraw-Hill, 2002
- [6] Gustiene, P., Gustas, R. (2002) "On a Problem of Ambiguity and Semantic Role Relativity in Conceptual Modelling", *Proceedings of International Conference on Advances in Infrastructure for e-Business, e-Education and e-Medicine on the Internet*, ISBN 88-85280-62-5, L'Aquila, Italy.
- [7] Negoro, F. (2001) "Methodology to Define Software in a Deterministic Manner", *Proceedings of ICII2001*, Beijing, China
- [8] Ackoff, R. L. (1989) "From Data to Wisdom: Presidential Address to ISGSR, June 88", *Journal of Applied System Analysis*, Vol. 16.
- [9] Turetken, O., Schuff, D. (2002) *The Use of Fisheye View Visualizations in Understanding Business Process*, *Proceedings of ECIS2002*, Gdansk, Poland

Chapter 7

Software Process Model and Configuration Management

This page intentionally left blank

The Lyee Based Process in Framework

Vincenzo Ambriola* Giovanni A. Cignoni* Hamido Fujita**

**Dipartimento di Informatica, Università di Pisa, Corso Italia, 40 – 56125 Pisa, ITALY*

***Iwate Prefectural University, 152-52 Sugo, Takizawa-Mura, Iwate, 020-0193, JAPAN*

Abstract. There is a general agreement in considering the development process as an important factor for the efficiency and effectiveness of software production. There are plenty of proposals of software development processes. The variety of proposals makes the comparison and the evaluation of these processes a strategic and intellectually rewarding activity.

This paper presents a novel and original framework for the comparison and the evaluation of software development processes. The proposed framework is applied to the Lyee Based Process, a process that relies on the Lyee technique to perform requirement analysis and on the Lyee tools for automated translation of requirements into executable code.

The main aim of the paper is to contribute to characterize the Lyee Based Process as a proposal comparable with other well known ones. The exercise of putting the Lyee Based Process in the framework also contributes to the definition of the development process based on Lyee, by highlighting its peculiar characteristics, and by identifying its weaknesses, as a contribute to its improvement.

1. Introduction

There is a general agreement in considering the process as an important factor that affects the efficiency and effectiveness of software production. Developers have the obvious interest in fulfilling the requirements in the most effective way and at the lower possible cost. Customers use the process as a source of confidence of paramount relevance in the software acquisition procedures. In this perspective, in fact, customers exploit their knowledge of the supplier's development process to apply stricter controlling actions.

There are plenty of proposals of software development processes. The production of training aids (such as books and courses) tools, and development environments targeted to a given process has become a profitable activity. Unified Software Process (USP) and Extreme Programming (XP) are the most notable examples in this context.

The variety of proposals available to the parties involved in software development makes the comparison and the evaluation of these processes a strategic and intellectually rewarding activity. Developers are focused on the elements that support the choice of the process to be used in their production process. Customers are interested in evaluating how properly suppliers perform their job.

The *Governmental Methodology for Software Providence* (Lyee, for short) is a software development context mainly based on the use of tools for the automatic translation of user requirements in executable code. The *Lyee Based Process* (LBP) is the software development process that is based on a custom requirement analysis and specification technique and on the usage of the set of Lyee tools.

The goal of this paper is to present a novel and original framework for the comparison and the evaluation of software development processes and to use such framework on the LBP to present its characteristics in a rational and open to discussion way.

The structure of the paper is the following: Section 2 presents how software processes can be classified in few categories. In Section 3, the activities of evaluating and comparing are described and related to the comparison framework. Section 4 presents the framework in general terms. Lyee & LBP are introduced in Section 5 while Section 6 is completely devoted to put LBP in framework. Conclusions and a reference section close the paper.

2. Categories of processes

There are many categories of “software processes”. Before addressing the problem of evaluating and comparing software processes one with each other, we need a precise definition of the concepts that we are investigating on. We can identify three classes of “entities”, commonly referred to as software processes.

- *Models*. A model is a taxonomy of the software development process. Its goal is to identify all the activities related to the software process, by defining them in terms of results and responsibilities, without imposing any constraint on their sequential organization. Examples of models are ISO/IEC 12207 [1] and the SPICE model of *base practices* [2] successively evolved in ISO/IEC TR 15504 [3].
- *Life-cycles*. A life-cycle is a high level organization of the development activities. Its goal is to identify the phases of the software process, by defining them in terms of dependencies, without entering into the detailed description of each activity. Historical, and always cited, examples are the *waterfall* [4] and the *spiral* [5] life-cycles.
- *Processes*. A process is a completely defined organization of the development activities, described at a fine level of details, that can be directly used as a model for instantiating the activities of a specific project for software development. Historical examples of processes are Cleanroom [6], PSS-05 [7], and more recent proposals such as USP [8] and XP [9].

As every classification, we do not claim its absolute value. For instance, it is worth to note that in the evolution of a specific proposal it is possible to move from one class to the other. In this transition, there are points in which a single proposal shows characteristics that belong to both classes.

To make concrete this observation, we recall the case of the standard developed and adopted by the European Space Agency (ESA). The previously cited PSS-05 was a typical *process*, characterized by very precise statements on the organization and the documentation of the development activities. The most recent ESA standards listed in the ECCS series [10, 11] show some aspects that are more related to *models*. The Agency has recognized a higher maturity of its suppliers and, therefore, has decided to impose standards that are expressed at a higher abstraction level. The effort of proposing their own *processes*, still conformant to the *model* prescribed by the Agency, has been left to the suppliers.

It is also worth to note that a process, to be a true process in our terminology, should be well defined and documented and stable in its definition. An undocumented or unstable process fails in being a valuable model for instantiating the activities of a specific software development project. It is possible to identify these requirements with the characteristics defined by level 3 of the Capability Maturity Model (CMM) [12] or ISO TR 15504. In this sense, we can say that these processes are mature.

In the follow of this paper, when we refer to software processes we always mean entities that clearly belong to the third class. Since these processes are proposed as organizational

models, directly usable in real life projects, we claim that they are the most interesting to be used as subjects of comparison.

3. Evaluating and comparing

Evaluation and comparison are two distinct activities. The goal of an evaluation is to express an absolute judgment; the goal of a comparison is to state a relative judgment. Evaluation can be applied to a single item; in order to perform a comparison we need at least two items.

The concept of maturity [13] is the main reference schema adopted for the evaluation of software development processes. Although interpreted in many different variations, maturity is practically the only concept that has been specialized for software development. As a matter of fact, the other widely applied schema is the ISO 9001 model [14], which is general and aimed at producing a conformity judgment.

Moving from CMM to ISO/IEC TR 15504 (still at the level of TR since 1998) there has been a general consensus on the idea that all the activities related to software development must be identified and evaluated. In particular, the use of a scale that measures the application of the best practices stemming from both the software engineering and the quality control traditions has been widely accepted. The outcomes of the *assessment* performed using CMM, or any other schema inspired from the maturity concept, are now widely used. Customers use assessments to qualify their suppliers. Suppliers, on the other hand, are more interested on the best practices suggested during the assessment activity as means to improve the efficiency and effectiveness of their processes.

Similarly, widespread recognized tools do not seem to exist in the field of comparison. Furthermore, the maturity assessment is more oriented to evaluate the implementation of a given process than to apply this evaluation to its own characteristics. The assessment methods based on CMM evaluate an organization by observing the whole set of the instances of its development software process. SPICE and, successively, ISO/IEC TR 15504 have better clarified the scope of an assessment explicitly saying that it is performed using data of projects already completed or still running on. By using assessment results, it is possible to derive an evaluation of the process itself. Nevertheless, the maturity concept is more suited to evaluate how a process is instantiated in a company.

Due to their dependency on the process instances, assessment methods based on the maturity concept are of little if not use in the choice of a process. Firstly, the process must be enacted in order to be evaluated. Secondly, although the evaluation is a qualitative judgment, it is not related to the process suitability within a given applicative domain or with respect to the size and the organization of the developer.

The aim of a comparison framework is, first of all, the identification of those characteristics that can be used to establish *process diversity* [15]. For instance, the framework can help in choosing the process that is best suited to the needs of a project or a company. In this context, the main goal is not a quest for the best process. We are searching the most suitable process and in this activity we cannot use any tool that implies the observation of process instances. We would be able to speculate on the process itself.

Following the approach successfully applied by many software product quality models (Boehm [16] and ISO/IEC 9126 [17] among the others) a comparison framework can be used as a *process quality model*, where quality is defined as the set of characteristics that make the process able to fulfill the user requirements.

Our framework, in fact, focuses on a set of features that can be the basis of the quality model. In this context, the main issue is the definition of metrics that can measure a process with respect to its quality characteristics. Since we are in a comparison framework we must

pay attention to the implicit constraint of evaluating quality in terms of differentiation and not with respect to an absolute scale.

4. The comparison framework

To develop a framework for comparing software processes we have to choose the process characteristics that are useful to establish process diversity. In other words, we are mainly interested in highlighting, in a given process, those peculiar issues that make the process itself an original proposal.

We start our analysis by remarking that many software development process proposals, in a naïve way, claim to have some peculiar characteristics of their own. Some examples of such characteristics are: efficiency with respect of costs or time, quality control, risk prevention and mitigation. We argue that this attitude is somehow influenced by marketing goals, especially when a software development process proposal comes together with books, training courses, production and management tools. Following this “salesman approach”, many proposed software development processes assume as proprietary and original some characteristics that actually should be recognized as shared among all processes. For this reason, in our proposed comparison framework we have not included such characteristics. We assume that these are common goals of the software engineering discipline: all process proposals should have these goals, but they must differ in how to accomplish them.

Having made this assumption we consider interesting for the framework the characteristics that allow us to characterize a process proposal with respect to the others. These characteristics should be useful in the task of deciding the suitability of the process to a given project or a given company. To identify these characteristics we refer to the reference schema usually adopted by *process modelling languages* (PMLs).

A process modelling language is designed having in mind the goal of describing any possible software process. To accomplish this goal the language should have a collection of semantic categories that naturally capture those aspects that are universally recognized as constituent elements of all processes [18]. Following this idea we built the comparison framework referring to *process elements* as the main items that have the property of identifying and characterizing a process proposal.

Characteristic	Code	Name	Description
Roles	R1	Role 1	...
	R2	Role 2	...
	
Phases	A1	Phase 1	...
	A1.1	Activity 1.1	... the role responsible is [Rx].
	A1.2	Activity 1.1	...
	...		
	A2	Phase 2	...
	...		
Products	P1	Product 1	... result of activity [An.m].
	...		
Tools	T1	Tool 1	... used for the making of product [Py].
	...		
Principles	1	Principle 1	...
	...		

In the last years the software process research community have proposed and discussed several PMLs. Among the others, we believe that [18] is a good survey on this topic. Since there is a common agreement on *roles*, *phases* and *activities*, *products* and *tools* as base process elements, we include them as the core part of our comparison framework. To complete the framework we added a new base characteristic, called *principles*, not previously listed in the PMLs context.

The discussion about PMLs has put in evidence other process elements that we have not included in the framework. For instance, humans are widely considered as a process element that a PML should manage. But humans cope with the instantiation of a process – humans play roles – not with the process definition. Because the framework aims to compare process definition and not process implementation, we decided not to include in the framework humans and other elements related to process enactment.

There are other process elements typical of PMLs that we decided not to include as characteristics of the framework such as, for instance, cooperation models or support to process evolution. The reason for excluding these characteristics from the framework is that they are more peculiar of the language than of the described processes. In this sense they are not interesting for our purpose.

The framework is sketched in Table 1 as a form to be compiled when the framework is applied to a given process. In the following we comment each characteristic of the framework by referring to the table to explain how it must be assessed when the framework is applied to a process.

- *Roles*. All process activities are performed by humans. Even in the case of an automated activity, there must always be a human that has the responsibility of the activity results. A role describes characteristics, skills, rights and responsibilities needed by a human to perform an activity or a set of activities. In the framework, roles must be uniquely identified and described. Conventionally, a role is coded as R_n , where n is a progressive number. Sometimes, to improve understandability, roles are hierarchically grouped (in this case n will be in the usual legal numbering format).
- *Phases and activities*. According to the usual definition (see for instance [19]), a process is a set of correlated activities that use resources to obtain a defined result. The framework requires the unique identification and description of all the development process activities and prescribes at least two abstraction levels: phases and activities. Conventionally, an activity is coded as A_n , where n is a progressive number for phases while for activities the number is in the usual legal numbering format. To improve understandability, activities can be decomposed in further levels. In each activity description, the role responsible for its results should be explicitly reported.
- *Products*. They identify the evident outcome of activities: a proper product must be persistent and identifiable. In practice it must be a printed document or a file under a reliable storage system – hopefully under version and configuration control. Products can be directly related either to the software application implementation (like, for instance, analysis documents, source code files, libraries and executables), or to the application documentation (like manuals and user guides) or to the documentation of activities (like project plans or test reports). Inside the framework products must be uniquely identified and described. Conventionally, a product is coded as P_n , where n is a progressive number. To improve understandability, products can be hierarchically grouped (in this case n will be in the legal numbering format). In each product description, there must be an explicit indication of the activity that outputs the product.

- **Tools.** The framework requires the unique identification and description of all the tools used in the software development process. The category includes software production tools as well as other tools like measurement tools or version and configuration management tools if their use is prescribed by the process. Conventionally a tool is coded as T_n , where n is a progressive number. If useful to improve understandability, tools can be hierarchically grouped (in this case n will be in the legal numbering form). In the description of each tool it should be noted the products for which the tool is used.
- **Principles.** This category includes basic management concepts, teamwork solutions, referred life cycle organization, and in general all software engineering best practices that characterize the process. Examples of principles are: pair programming and test-design-first (like in XP), strict separation of coding and testing activities (Cleanroom), full documentation of activities (PSS-05), adoption of analysis and design languages (USP). The framework requires the identification and the description of the principles that can be recognized in or that inspired the process. Principles are used in the framework as a mean to characterize the process. As they are not strictly related to the other characteristics there is no need to code them (they are just sequentially numbered).

Application and customer	Dimension	Composition of the system
401K operation and management system Major Japanese insurance company	2800 KL developed 2000 KL in use COBOL	Web online (Hitachi TP1/Web), Oracle®
Sales/purchase order management system Major Japanese insurance company	1200 KL COBOL	Web online (Hitachi TP1/Web), Oracle®
Sales/purchase order management system Major Japanese insurance company	200 KL COBOL	IBM® host computer, Client-server, Oracle®
Procurement management system Major Japanese heavy industry manufacturer	260 KL Java™	Web online (WebSphere®), SQL server
Procurement management system Major Japanese heavy industry manufacturer	240 KL Java™	Web online (WebSphere®), SQL server
Production management systems, #1 and #2 Major Japanese heavy industry manufacturer	40 KL each system Java™	Web online (WebSphere®), SQL server
Cost accounting system Major Japanese steel manufacturer	300 KL COBOL	IMB® host computer, Client-server, DB2®
Securities management system Major Japanese commercial bank	100 KL Visual Basic®	PC stand-alone, Microsoft® Access
Software system to support the user's business Major Japanese retailer with a nationwide network	Being developed now RPG + Java™	Web online (WebSphere®), DB2®
Sales management system Catena Corp.	700 KL COBOL + RPG	AS/400® host computer, DB2®
Fixed asset ledger management system Catena Corp.	150 KL COBOL	AS/400® host computer, DB2®
Customer record management system Affiliated company of Catena Corp.	500 L C	WindowsNT®, Client-server, Oracle®
Storenet®, package for HQ of retailing chains (being sold by Catena Corp.)	300 KL C	WindowsNT®, Client-server, Oracle®
ToReBiS®, package for HQ of restaurant chains (being sold by Catena Corp.)	300 KL Visual Basic®	Web online (Microsoft® IIS), Oracle®

Putting a process in the framework is an assessing activity. Before performing an assessment, the process must be clearly identified in terms of the documentation that will be used as a reference to identify its characteristics. During the assessment it is important to rely only on the documentation: any assumption made in this activity will compromise the reliability of the result. When the process is not well defined (i.e. it does not possess the maturity property that we have discussed in Section 2), it is possible to find mismatches or omissions in the description of a characteristic. For instance, a common flaw is the use in the process of a product that is not related to an explicitly defined activity. In this sense, discovering a process flaw is itself a result of the assessment, useful to identify those parts of the process that are not well defined or that require a custom implementation when the process is adopted in a real project.

The framework can be easily used for comparing two processes: both processes are independently inserted in the framework and the results are compared. A rough but effective way to establish a process complexity is to count the number of items used in its description. This figure can be related to the effort needed for its adoption in a company. Another useful information comes from the comparison of the discovered process flaws.

At a second level it is possible to find similarities in the items definition. For instance, it is common that roles have different names although they identify the same responsibilities and require the same skills. The similarity check can be performed on activities and products, with respect to their relations with other characteristics. For instance, two activities are similar if their descriptions are similar and if a similarity can be found in the roles and the products they are related to.

5. Lyee and the Estate project

Lyee [20] is a methodology for software development that is mainly based on the use of a peculiar way to elicit and specify the requirements and then on the adoption of tools for the automatic translation of such requirements in executable code.

Lyee was defined by Mr. Fumio Negoro, president of the *Institute of Computer Based Software Methodology and Technology* (ICBSMT), which is now in charge of the research about Lyee, its improvement, development of related tools, and training. The Lyee method is in use at Catena Corp. (Catena) and at other companies for the implementation of business software applications. In Table 2 are reported some of the software projects carried out using the Lyee methodology. Dimensions are only estimated and are expressed in kilo lines of source code (KL).

A preliminary analysis of the development process based on Lyee suggested the presence of intrinsic differences between the LBP and the development processes usually adopted in Europe and in the US. This difference seems to exist also with respect to recent and non-traditional development approaches (such as the RUP or XP).

Following this first impression the *Estate* project [21] was started in July 2001 as part of the Lyee International Collaborative Research Project. The Estate research project aims to investigate and document the implications of the Lyee adoption on the software development process of Catena. The first goal of Estate will be the comparison of the process based on Lyee with other approaches, contributing to the discussion on the software process and the related development methodologies. The second goal will be to help Catena in the better understanding of its development process and in the finding of suggestions for the improvement of the process and for the further development of the Lyee method.

6. LBP in framework

Before attempting to put the LPB in the comparison framework we have to answer a fundamental question: is the LBP a process according to the definition given in Section 2? In other words, have the research around Lyee and the practice carried out by Lyee users developed and documented a mature process?

The relevance of this issue is twofold. On one hand the framework requires processes that have reached a stable and mature definition. On the other hand, if a process cannot be directly used as a model for instantiating the activities of a specific software development project, it is not fair – and not very useful – to compare it with mature competitors.

Regarding LBP, the answer is no. There is no clear evidence of a mature and documented LBP. However, it seems that the use of the Lyee tools and the requirements elicitation activities carried out by Catena and ICBSMT have established a praxis. Being aware of this issue, we anyway decided to try to put LBP in the framework. As a reference we used our knowledge of the praxis, as it was perceived by interviews and other contacts with people directly involved in projects in which Lyee is used.

In this perspective we used the framework not to describe an already mature process but to understand a praxis, to verify whether this praxis has something original that makes it different from other process proposal, and, in the case, to help to move such praxis toward a well defined process. In this case the framework can be used as a tool to verify the process and discover its flaws. While flaws are a serious problem for a commercial proposal, they are interesting results for a process that is still in development.

In the following we analyze the characteristics of the LBP. As the framework prescribes we start from roles and then proceed with phases and activities, products, tools and principles. For each characteristics the results of our assessment are presented in a table and briefly discussed.

Code	Name	Description
R1	Customer roles	These roles are played by people on the part of the customer who interact and are supported by the developer roles.
R1.1	Domain expert	Expert user who specifies the domain of the application.
R1.2	Cutomer representative	Represents the customer for all the decisions regarding the acceptance of the application.
R2	Developer roles	These roles are played by people on the part of the developer.
R2.1	Analyst	Specifies the accurate PRD, and the screen transition diagram for the needed requirement.
R2.2	Coder	Uses the Lyee tools to generate the code; this role does not perform any actual programming task and is a very low skilled role.
R2.3	BS designer	Designs the boundary software components using conventional methodologies. This role is a technical senior role.
R2.4	BS programmer	This role is in charge of the programming activities for the implementation of the boundary software components. This role is a technical junior role.
R2.5	Core code programmer	This role is in charge of the programming activities needed to rework the automatically generated code to prepare it for the integration with the boundary software components. This role is a technical junior role.

Table 3. The Lyee Based Process in the framework: roles

According to the framework assessment technique, Table 3 describes the roles we found in our investigation of the praxis established in the project carried out using the Lyee methodology. We were able to find seven well defined roles, classified in two groups: *customer roles* and *developer roles*.

The classification of the roles in two distinct groups is a noticeable characterization of the LBP. In fact the LBP assumes as a main issue the involvement of the customer. In particular several activities (A1.1, A1.2, A1.3, see Table 4) are responsibility of the customer, the developer is involved in these activities, but as technical support. We will remark these characteristics in the framework section devoted to principles.

The identified roles on the part of the customer comprehend several technical roles. It is a characteristic of LBP – at least in the praxis we observed – that management roles are not defined. There is obviously a project responsible, but how this role is related to the activities of the development process is not well defined (we can say that for this aspect they are working at CMM level 1). In the future development of LBP it will be possible to decide to define the management activities or to explicitly decide to leave them out of the process.

Code	Name	Description
A1	External design	A typical analysis phase in which the “external” attributes highlight the strong involvement of the customer.
A1.1	Screen transition analysis	The outline of the system specification design. The role responsible is [R1.1] supported by [R2.1].
A1.2	Specification of words	Words can be seen as individual memory elements (or variables) whose purpose is to record data and specify how and when these data are valued, input or output. The role responsible is [R1.1] supported by [R2.1].
A1.3	Screen/printout layout analysis	This result represents the confirmation needed by the user to check the requirement analysis. The role responsible is [R1.1] supported by [R2.1].
A1.4	Logical DB definition	Logical schema for the application DB. The role responsible is [R2.1], while [R1.1] is involved.
A2	Internal design	A typical design phase in which the “internal” attributes highlight that activities are carried out without the involvement of the customer. Lyee tools are used to automate some activities.
A2.1	Process route diagram definition	In Lyee it is the horizontal representation for all the requirements. The role responsible is not defined.
A2.2	Word list generation	Activity carried out by using LyeeBelt. The role responsible is [R2.1].
A2.3	Screen/printout definition	Definition of all application screens and printouts. The role responsible is [R2.1].
A2.4	Physical DB definition	Physical schema for the application DB. The role responsible is [R2.3].
A2.5	Test data definition	Definition of the validation test suite for the application. The role responsible of its development is not defined.
A3	Implementation	A typical implementation phase, mainly carried out by automatic code generation.
A3.1	Core code generation	Activity carried out by using LyeeAll. The role responsible is [R2.2].
A3.2	Reworking of core code	Programming activity to prepare the automatically generated code for the integration with the BS components. The role responsible is [R2.5].

A3.3	BS development	Analysis, design, coding and component testing activities for the development of all the BS components. The role responsible is [R2.3] and [R2.4] are used as working resources.
A3.4	Integration	Integration and verification activities for the development of all the BS components. The role responsible is [R2.3] and [R2.4] are used as working resources.
A3.5	Trial test	Validation activity on the integrated application. The role responsible is not defined.
A4	Application test	Final validation and formal acceptance of the application.
A4.1	Final application test	This is actually the last instance of the A3.5 activity.
A4.2	Acceptance	Formal acceptance. The role responsible is [R1.2].

Table 4. The Lyee Based Process in the framework: phases and activities

From a different point of view, yet regarding the developer roles, it is worth to note that there are different roles related to programming activities. In this case skills and competencies establish the difference between roles: there are roles for the activities related to the Lyee methodology and roles related to conventional software development techniques needed for building and integrating the boundary software components.

Code	Name	Description
P1	Analysis results	These are the analysis in regard to the development done using Lyee methodology
P1.1	Process route diagram	The PRD specifies the navigation between the various components of the application. It is a result of [A1.1, A2.1].
P1.2	Word list	Words can be seen as individual memory elements (or variables) whose purpose is to record data and to specify how and when these data are valued, input or output. The word list is a result of [A1.2, A2.2].
P1.3	Screens	User defined interfaces between the words and the application, results of [A1.3, A2.3].
P1.4	Scenario functions	The SF implements a basic behavior of an application components. Scenario Functions are results of [A1.3, A2.5].
P1.5	DB schema	This is the logical DB definition, result of [A1.4].
P2	Core code	This group comprehends all the code that is mainly developed using the Lyee methodology.
P2.1	Lyee generated code	This is the automatically generated code obtained as output of the Lyee tools [A3.1].
P2.2	Reworked generated code	This is the Lyee generated code after it is manually reworked [A3.2] to prepare it for the integration with the boundary software components.
P3	Boundary software components	This group comprehends all the components that are needed to build the application but are not developed using the Lyee methodology (also referred as "not-Lyee components"). Products in this set include design documents, code and libraries.
P3.1	Physical DB	Implementation of the DB, result of [A2.4].
P3.2	Other needed "not-Lyee" components	Products in this set include design documents, code and libraries of all "not-Lyee" components other than the DB. They result from [A3.3].

P4	Deliverable application	This is the main product of the development process. It is the result of the integration of the reworked generated code with the boundary software components [A3.4].
P5	Application documentation	Product documentation.
P5.1	User manual	Documentation for the end user. There is no activity defined to produce this output.
P5.2	Maintenance guidelines	Documentation for application administrators. There is no activity defined to produce this output.
P6	Reports	Process documentation.
P6.1	Acceptance report	Report at the end of each trial (validation) test [A3.5].
P6.2	Application acceptance report	Report at the end of the last validation test [A4.1, A4.2] that formally accepts the application.

Table 5. The Lyee Based Process in the framework: products

Table 4 describes phases and activities resulting from our assessment of the LBP. Several remarks are worth to note. The most important one is that, while not explicitly stated in the process definition, the LBP is an evolutionary life cycle. The phase's external design, internal design and implementation are iterated until the validation activity made at the end of the implementation phase succeeds.

The LBP assumes that the application is based on a DB, in this sense the LBP is a process oriented to typical business applications. With *boundary software* (BS) they define the basic components needed to integrate the Lyee generated code with the environment or the components that is not convenient to develop using Lyee (for instance because they are implemented with reused code). In this perspective, the explicitly defined activities for building the BS should represent a little part of the overall application development effort. This can be considered an explanation for collapsing all analysis, design coding and component testing activities for all the BS development in only one activity (A3.3), while, for instance, the DB is identified as a single component and there are explicitly separated analysis and implementation activities (A1.4, A2.4).

The definition of the testing activities in the actual LBP praxis needs improvement. In particular, no responsible role is in charge of the definition of the validation test suite for the application. A possible reason is that these activities involve the customer; this can also explain the lack of a product (there is no related product in Table 5), but the existence of at least a document should be stated.

Other notes about the assessment results about product shown in Table 4, regard the application documentation, which is defined, but there are no activities that produce it, and the acceptance reports, which are the only proper process documentation, i.e. document not directly related to the developed application but to the development process. As already stated, the LBP is not yet mature about management issues.

Code	Name	Description
T1	Lyee tools	These are tools that are peculiar of the LBP and are based on the Lyee methodology for requirement analysis and code generation.
T1.1	LyeeAll	Used for the generation of [P2.1]. It is a commercial Case environment that transforms Lyee software requirements into code.
T1.2	LyeeBelt	Used for the generation of [P1.2]. Transforming the requirement into a sort of pallets, using the LyeeBelt. Pallets are composed of <i>words</i> grouped into <i>logical units</i> .

T3	Office automation tools	Tools used to prepare the input files [P1.1, P1.3, P1.4] for the Lyee tools.
T4	Core language tools	Tools used to rework the code generated by LyeeAll to obtain [P2.2] and to integrate it with the boundary software components in the deliverable application [P4].
T5	Boundary software tools	Tools used to build the boundary software components [P3].

Table 6. The Lyee Based Process in the framework: tools

Regarding LBP tools, as reported in Table 6, the most specific ones are the Lyee tools for the automatic generation of code. As reported in the principles we elicited the LBP is based on the use of a methodology for the definition of application requirements and the use of tools for the translation of these requirements in executable code. To complete the application the generated code must be integrated with the BS components: the other tools involved are therefore conventional software development tools used for building BS and for integration activities. BS tools may coincide with the core language tools, i.e. the tools needed for reworking the automatically generated code for integration, but generally speaking they depend on the context in which the application has to be developed.

Code	Name	Description
1	Strong customer involvement	The customer is explicitly involved in the development process. The process includes activities that are responsibility of the customer.
2	Automatic code generation	The process relies on the adoption of a requirement analysis technique and on the use of tools for the automated translation of requirements in the application.
3	Evolutionary life cycle	The analysis/code generation/validation activities (phases A1, A2 and A3) are iterated until the customer validates and accepts the application.

Table 7. The Lyee Based Process in the framework: principles

The last identified characteristics of the LBP, reported in Table 7, are the principles that it is possible to recognize in the process definition. As previously anticipated, in our assessment of the LBP we identified at least three widely known principles: strong customer involvement, automatic code generation and evolutionary life cycle.

It is possible to identify also the test-design-first principle, because there is an activity (namely A2.5) devoted to the definition of the validation test suite for the application, and this activity is carried out before the Implementation phase. However, all the activities related to verification and validation are not well identified in the LBP at this moment, so it is difficult to actually establish if this principle is in LBP or it is suitable to be in a more mature version of the process.

7. Conclusions

In this paper we presented a framework for comparison of software processes. We also give a definition that identifies the software processes suitable to be compared, i.e. the ones able to supply a completely defined organization of the development activities that can be directly used as a model for instantiating the activities of a specific project. Our perception

is that, given the many proposals, even commercial, of software processes, it is important to develop a framework that helps in the evaluation of which processes are best suitable for a company or a software project.

The proposed framework permits the assessment of a process with respect to a given grid of characteristics. The characteristics are those items that the research in process modeling languages identified as basic in the definition of a software process. In this perspective, we used the same process languages tools to analyze a process, to discover the presence of flaws and to identify its peculiar characteristics. The assessment with respect to the comparison framework can be considered the rationale of the high level analysis performed when a process is described in a process language.

In the second part of the paper we applied the framework to the LBP, the software development process based on the Lyee methodology. Being the comparison framework still object of research, we exploited this opportunity both to experiment with the framework and to help in the development of the LBP.

The results of the assessment we made to put the LBP in the framework show that the comparison framework is a good tool for the analysis of processes. With relatively low effort we were able to give a fair description of the LBP – at the moment this is the only documented LBP description available – that both describes the LBP peculiarities and gives some hints to drive the future development of the process. Here is a brief summary of the most relevant issues we discovered about LBP:

- the LBP can be considered different, i.e it is a new proposal and not just the introduction of some tools in a typical software process;
- the LBP diversity consists in the adoption of several software engineering principles and in the use of a methodology that mostly drives the requirement analysis;
- the evolutionary nature of the life cycle is not made explicit, this must be solved in the future development of the LBP; in particular the activities needed to plan and manage the evolutionary loops must be included in the process;
- several activities like verification during integration, validation of the application, and product documentation are not well defined.

Software production is a complex process, the development of new technologies and the availability of new hardware platforms increase the demand of reliable software applications. Both software supplier and software customers look at the software development process to something that is crucial for quality assurance. A new development tool or a new software engineering methodology are not perceived by software producers as viable innovations if they are not completed with a process that envelop them in a ready-for-use set. The success of the UML/USP pair is one of the better examples of the value that a defined process adds to a set of tools or methods.

In this context the comparison framework can be useful from two different perspectives. First of all, it is a tool for identifying and comparing the characteristics of different software process proposals, helping in the choice of the better suitable process for a given company or a given projects.

On a different perspective, as in the case of LBP, the framework can be used to assess a praxis that comes out from the use of a new development methodology – Lyee, in our case – and to help in the definition of a software process that adds value and completes the proposal of the methodology to the companies.

8. References

- [1] International Organization for Standardization, "ISO/IEC 12207:1995 Information Technology – Software lifecycle processes", 1995.
- [2] SPICE Document Suite, available at <http://www-sqi.cit.gu.edu.au/spice>, accessed February 2002.

- [3] International Organization for Standardization, "ISO/IEC TR 15504:1998 Information technology – Software process assessment — Parts 1-9", 1998.
- [4] W.W. Royce, "Managing the development of large software systems: concepts and techniques", Proceedings of *Wescon '70*, August 1970.
- [5] B.W. Boehm, "A spiral model of software development and enhancement", *IEEE Software*, May 1998.
- [6] H.D. Mills, M. Dyer, R.C. Linger. "Cleanroom software engineering", *IEEE Software*, September 1987.
- [7] European Space Agency, "PSS-05 Issue 2 – Software engineering standards", 1991.
- [8] I. Jacobson, G. Booch, J. Rumbaugh, *The Unified Software Development Process*, Addison-Wesley, 1999.
- [9] K. Beck, *Extreme Programming Explained*, Addison-Wesley, 2000.
- [10] European Space Agency, "ECCS-40a – Space engineering: software", 1999.
- [11] European Space Agency, "ECCS-80a – Space product assurance: software product assurance", 1996.
- [12] M.C. Paulk, et al., "Capability Maturity Model for Software, Version 1.1", Software Engineering Institute, CMU/SEI-93-TR-24, 1993.
- [13] W.S. Humphrey, *Managing the software process*, Addison-Wesley, 1989.
- [14] International Organization for Standardization, "ISO 9001:2000 – Quality management systems – Requirements", 2000.
- [15] *Process Diversity*, cover story, *IEEE Software*, Vol. 17, No. 4, July/August 2000.
- [16] B.W. Boehm and others, *Characteristics of Software Quality*, North Holland, 1978.
- [17] International Organization for Standardization, "ISO/IEC 9126:2001 – Software engineering – Product quality – Part 1: Quality model", 2001.
- [18] R. Conradi, M.L. Jaccheri, Eds., "Process Modelling Languages", in "Software Process: Principles, Methodology, and Technology", *Lecture Notes in Computer Science*, No. 1500, Springer-Verlag, New York, 1998.
- [19] International Organization for Standardization, "ISO 9001:2000 – Quality management systems – Fundamentals and vocabulary", 2000.
- [20] Lyee Web site, <http://www.lyee.co.jp/>, accessed June 2002.
- [21] Estate Project Web site, <http://www.di.unipi.it/estate/>, accessed June 2002.

Configuration Management Concept for Lyee Software

Volker GRUHN, Raschid IJIOUI, Dirk PETERS, Clemens SCHÄFER
Fachbereich Informatik Lehrstuhl 10,
Universität Dortmund, D-44221 Dortmund, Germany

Abstract.

A configuration management concept is presented for software projects using Lyee methodology. To show this concept an introduction in configuration management is given. Then, the structure of Lyee programs is redefined by sets and their dependencies. From this redefined structure, the actual configuration management concept is derived and discussed.

1 Introduction

Software configuration management has been defined as the discipline of controlling the evolution of complex software systems [12]. As a vital task for professional software development, it also has to be provided for software projects carried out using Lyee methodology. In this paper we discuss a software configuration management concept for Lyee.

First, we give a short introduction into configuration management in general and software configuration management in particular. Additionally, we describe an already existing software configuration management testbed as base for our implementation. Then, we re-define the structure of Lyee programs using mathematical definitions, in order to provide a sound basis for further discussions. These definitions make it possible to map the entities of Lyee programs onto configuration items introduced before. Next, we depict the architecture of a system using the testbed. Finally, we discuss the concept and how its realization and evaluation will take place.

2 Configuration Management and Software Configuration Management

2.1 Configuration Management

A standard definition of configuration management can be found at [7, 8, 4]. It describes the following configuration management procedures:

- *Identification.* Reflects the structure of the product, identifies components and their type, making them unique and accessible in some form¹.

¹The IEEE software configuration management standards [7, 8] denote components by configuration items; the synonyms configuration object or simply object are also found.

- *Control*. Controls the release of a product and the changes to it throughout its life cycle. This is done by having controls in place that ensure consistent software via the creation of a baseline product.
- *Status Accounting*. Records and reports the status of components and change requests, and gathers vital statistics about components in the product.
- *Audit and Review*. Validates the completeness of a product and maintains consistency among the components, ensuring that the product is a well-defined collection of components.

Further surveys on software configuration management [4] extend this definition to include procedures like construction management, process management, and team work control:

- *Manufacture*. Manages the construction and building of the product in an optimal manner.
- *Process Management*. Ensures the carrying out of the organization's procedures, policies and life cycle model.
- *Team Work*. Controls the work and interactions between multiple developers.

2.2 Software Configuration Management

In [5] an approach to classify software configuration management functionality is made. According to this examination the following two major software configuration management approaches can be identified:²

- *Checkin/Checkout Model*. The basic software configuration management model introduces the concept of a repository holding multiple versions of a product component. Developers can copy versions from (check out) and to (check in) the repository.
- *Change-Oriented Model*. As its name says, the change-oriented model focuses on changes rather than on versions. In this model, versions are the product of change set applied to a baseline. This model is useful for propagating and combining changes across users and sites.

To discuss the software configuration management concept for Lyee properly we introduce some more definitions to make the key role players in the software configuration management game explicit (cited from [2]):

- A (*software*) *object* (item) is any kind of identifiable entity put under software configuration management control. An object may be either *atomic*, i.e. it is not decomposed any further (internals are irrelevant to software configuration management), or *composite*. A composite object is related to its components by *composition relationships*. Furthermore there may be *dependency relationships* between dependant and master objects.

²In [5] the Composition Model and the Long Transaction Model are also mentioned. These two are left out here due to their minor relevance

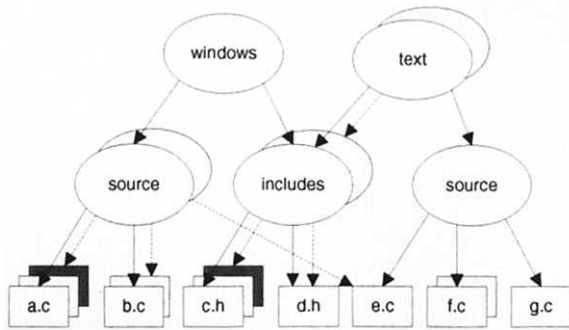


Figure 1: Sample NUCM Repository

- A *version* is an implicit or explicit instance of a versioned object which groups “similar” objects sharing same properties. We distinguish between *revisions* (historical versioning) and *variants* (parallel versions).
- A *configuration* is a consistent and complete version of a composite object, i.e. a set of component versions and their relationships. A configurations is called *bound* if it exclusively contains versions as its components. Conversely, an *unbound* configuration exclusively consists of versioned objects. A *partly bound* configuration lies in between.

According to [3] a last definition has to be set up:

- A *delta* is the difference between two versions and serves to identify the changes and to save space in the repository.

3 The NUCM System

Many research activities have been performed on the topic of software configuration management. Thus, many configuration management tools have evolved from these efforts. For this reason we tried to find an existing software configuration management tool that is suitable for our software configuration management concept for Lyee software.

After the comparison of different software configuration management systems³ where we examined their flexibility, the availability of source code, and the suitability for being adapted we came to the conclusion to use the Network-Unified Configuration Management (NUCM) introduced in [6].

NUCMs storage model allows the user to store and to version *artifacts*. Artifacts refer to the software objects mentioned above. As for software objects, artifacts can be atomic or composite. Atomic software objects are called *atoms* within NUCM. Composite software objects are called *collections* within the terminology of NUCM.

³In particular, we compared the Unified Software Versioning Model (USVM, see [14]), the Uniform Version Model (UVM, see [13]), the Concurrent Versions System (CVS, see [1]), the Resource Control System (RCS, see [11]) and NUCM.

An atom can be shared among multiple collections. Collections can be used recursively; they can be part of or larger, higher-level collections. A repository consists of one or more top-level collections which serve as the entry points to the contents of the repository.

Figure 1 presents an example to illustrate the basic concepts of NUCM (see [6]). The figure shows a portion of a repository for the C code of a hypothetical software. However, NUCM is not restricted to C code software. This example has been chosen to show the storage of dependent objects in the NUCM storage. Collections are shown as ovals, atoms as rectangles. Containment relationships are indicated as arrows. Two top level collections, windows and text contain separate collections called source and a shared collection called includes. Both the collections source and includes only contain atoms that are files in this example. Versioning is indicated by the use of shades. A darker shade represents an older version of an artifact. Dashed lines indicate containment relationships for older versions of collections. For example, the older version of collection includes contains an older version of atom c.h. Both the older and newer versions contain the same version of atom d.h.

NUCM provides functions that allow the creation of views of repository contents. That means, the contents of the repository are extracted to a file system. This file system will have the same structure as the data in the repository. Other functions that read the contents of a file system and store the data in the repository are provided as well.

4 Static Structure of Lyee Software

The definitions of the key role players of a software configuration management system depicted above introduced entities as software objects or composition relationships. For setting up a software configuration management concept for Lyee, the elements of Lyee programs need to be mapped onto these configuration items. This implies, that the relationships between the Lyee program items are clearly defined.

One problem we were faced with is the fact that it is hard to get a formal view of the structure of Lyee software from other publications. Thus, we decided to generate mathematical definitions to illustrate the relations between the entities of Lyee software.

For these considerations we modeled the parts of Lyee programs that are of structural interest. The aim of these definitions is to capture the static structure of Lyee programs. It is not intended to provide a dynamic model that could represent a Lyee program at runtime. The definitions have been derived from the description of the Lyee structure in [9, 10] and from the study of the database structure of the LyeeALL tool in combination with information from the Institute of Computer Based Software Methodology and Technology, Tokyo. The study of the database structure of the LyeeALL tool enhanced our understanding of the Lyee program structure and inspired the definitions below.

As we tried to emphasize the parts of Lyee structures that are of structural relevance for our software configuration management purpose, the definitions are incomplete to a certain extent: in all cases there are additional attributes. We left out these attributes to keep the definitions simple. This is possible as the attributes are not necessary for the understanding of the program structure. All these attributes have been subsumed in the X_i that can be found in the definitions. We begin with the topmost entity that can be found, the project.

Definition 1 (Project) A Lyee project is a tuple $j = (n_j, S, X_j)$ consisting of the *name* of the project n_j and a set of systems $S = \{s_1, \dots, s_x\}$ with $x \in N$. \square

Regarding the information gathered from the examination of the LyeeALL tool, a project can be decomposed into one or multiple systems.

Definition 2 (System) A Lyee system is a tuple $s = (n_s, D, F, X_s)$ with n_s as the name of the system. $D = \{d_1, \dots, d_x\}$ is a set of *process route diagrams*, $F = \{f_1, \dots, f_y\}$ is a set of *defineds* with $x, y \in N$. \square

The process route diagrams in the system are discussed later. We continue with the definition of the defineds.

Definition 3 (Defined) A *defined* in terms of Lyee theory can be specified as a tuple $f = (n_f, t_f, X_f)$, where n_f is the *name* of the defined, t_f is the *type* of the defined. Typically, the type of the defined t_f will be one of $\{screen, database, \dots\}$. \square

A defined refers to entities as database tables or dialog windows. The names of these entities will be reflected by n_f . X_f stands for additional technical information that has to be stored software configuration management system but that has no structural information in it. The type of the defined t_f was left separate in the definition as for configuration management purposes this type indicates which additional data not covered by the LyeeALL tool has to be processed. For example, in case of $t_f = screen$ the definition of the related dialog window has to be put under software configuration management control, too. We continue with the definition of the items in a defined.

Definition 4 (Item in Defined) An *item in a defined* can be specified as tuple $i = (n_i, f, X_i)$ where n_i is the *name* of the item in the defined, f the defined. \square

The items of a defined are e.g. field names within a database table or dialog fields within a dialog window.

As mentioned above, the second major element of systems besides the defineds are the process route diagrams. We continue with their definition.

Definition 5 (Process Route Diagram) A *process route diagram* is defined as tuple $d = (n_d, B, b_0, p_0, X_d)$ where n_d is the *name* of the process route diagram. $B = \{b_1, \dots, b_x\}$, $x \in N$ is a set of *base structures*. $b_0 \in B$ is an *initial base structure*, p_0 an *initial pallet*. \square

Closely related to the definition of the process route diagram is the base structure.

Definition 6 (Base Structure) A *base structure* is a tuple $b = (n_b, b_p, p_4, p_2, p_3, X_b)$ where n_b is the *name* of the base structure, b_p is the *parent base structure*. p_4, p_2 and p_3 are pallets. If a base structure has no parent, we write $b_p = \emptyset$. \square

Here as well, additional information as e.g. the interaction type can be subsumed in X_b . If we continue with the refinement of our definitions, we have to define the pallets.

Definition 7 (Pallet) A pallet is a tuple $p = (n_p, L, R, X_p)$ consisting of a set of *logical units* $L = \{l_1, \dots, l_x\}$ and a set of *routing vectors* $R = \{r_1, \dots, r_y\}$ with $x, y \in \mathbb{N}$. n_p is the name of the pallet. \square

The next items to be defined are the logical units.

Definition 8 (Logical Unit) A logical unit is a tuple $l = (n_l, f, t_l, S, A, X_l)$ where n_l is the name of the logical unit, $t_l \in \{\text{input}, \text{output}\}$ is the type of the logical unit, f is the defined the logical unit is assigned to, $S = \{s_1, \dots, s_x\}$ is a set of *signification vectors*, $A = \{a_1, \dots, a_y\}$ is a set of *action vectors* with $x, y \in \mathbb{N}$. \square

The logical units lead to the definitions of the signification vectors and the action vectors.

Definition 9 (Signification Vector) A signification vector in terms of Lyee methodology is a tuple $v_s = (n_{v_s}, w, W_{dom}, Q, X_{v_s})$. n_{v_s} is the name of the signification vector, w is the word that is calculated by the signification vector, Q is a set of functions that are executed to calculate w . This refers to the contents of the up to seven boxes of a signification vector. Let φ be the second element of Q so that φ represents the program code stored in the second box of the signification vector. W_{dom} are all words in the domain of φ , this means all words that are necessary to execute φ . When signification is established according to Lyee terminology, $\varphi(W_{dom})$ is calculated and the result is stored in the word w . \square

Definition 10 (Action Vector) An action vector is a tuple $v_a = (n_{v_a}, w, l, t_{v_a}, Q, X_{v_a})$ with a word w and a logical unit l . n_{v_a} is the name of the action vector, Q contains the contents of the seven boxes and thus represents the requirements in Lyee terminology, $t_{v_a} \in \{\text{input}, \text{output}, \text{structural}\}$ is the type of the action vector. \square

This definition of the action vector captures the input vectors, the output vectors and the structural vectors in terms of Lyee methodology. The remaining vector type not covered yet is the routing vector. We define:

Definition 11 (Routing Vector) A routing vector is a tuple $v_r = (n_{v_r}, w, t_{v_r}, p, X_{v_r})$ with the name of the vector n_{v_r} , the word w , the type $t_{v_r} \in \{\text{duplex}, \text{continuous}, \text{multiplex}, \text{recursive}\}$, and the pallet p . \square

Although the type of the routing vector has no implications on the static structure of the routing vector within the Lyee program and thus could be subsumed in the X_{v_r} , we modeled this information separately in the t_{v_r} . This is due to the reason that the type of routing has implications on the structure of the software as whole and has to be regarded when a component oriented view on Lyee software is established.

Definition 12 (Word) A word in terms of Lyee theory is a tuple $w = (n_w, X)$ with n_w as the name of the word. \square

Words are used by different vectors. Due to the definition of these vectors, words are always related to a pallet. Words can also be related to a logical unit, which is not necessary. This leads to the consequence that one word representing a certain data entity will be reflected by multiple words of the type defined here. However, these multiple words will have the same name, the same logical unit but differ in the pallet.

Definition 13 (Boundary Word) A *boundary word* in terms of Lyee theory is a tuple $w_b = (n_{w_b}, w, X_{w_b})$ with n_{w_b} as the *name* of the boundary word and the *word* w . The boundary word w_b can be regarded as a reference to the word w . □

The definition of the boundary word is shown here for completeness. With these definitions we redefined the static structure of Lyee programs in a way to formulate the actual configuration management concept.

5 Lyee Configuration Management Concept

The configuration management concept for Lyee software can be split up into two tasks: first, a mapping has to be defined that associates the elements of the static structure of Lyee programs with software objects in the NUCM system. Then, the operations to put a Lyee program under software configuration management control or to retrieve a version of the program from the repository have to be specified.

5.1 Mapping the Lyee Program Structure onto Software Objects

Based on the definitions of the Lyee program structure, the following mapping rules can be set up:

- A tuple always becomes a collection in the NUCM repository.
- Any item of a tuple in a definition that is neither a set nor a tuple will become an atom in the NUCM repository.
- Any X_ν in a definition will become a collection in the NUCM repository. In this collection, all items forming the X_ν will become atoms.
- All sets within tuples become collections in the NUCM repository.

Figure 2 shows the NUCM notation of a pallet and a logical unit. The tuple of the pallet is transformed into the top-level collection p . Its element correspond to the definition of the pallet $p = (n_p, L, R, X_p)$: the name n_p is stored as atom. L is the set of logical units and thus stored as collection. According to the rules, The set of routing vectors R and X_p are stored as collection, too. L consists of three logical units $\{l_1, l_2, l_3\}$ in this example. The refinement is shown for l_2 .

The elements of the collection l_2 are set up according to the definition of the logical unit $l = (n_l, f, t_l, S, A, X_l)$. The name n_l and the type t_l are stored as atoms, the defined f , the sets S and A , and the additional data X_f are stored as collections. Note that the elements of some collections are not drawn in this example.

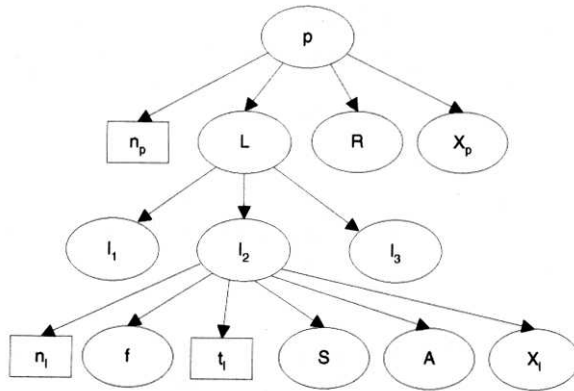


Figure 2: Pallet and Logical Unit as NUCM Artifacts

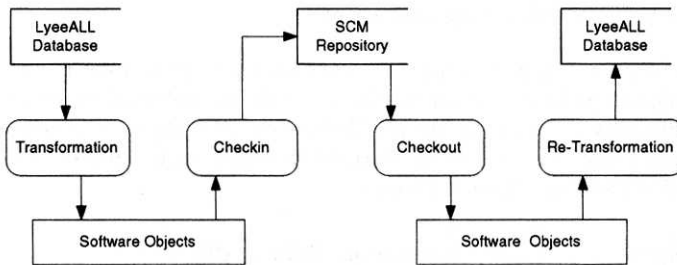


Figure 3: SCM Information Flow

5.2 Configuration Management Structure

With the rules set up in the previous section it is possible to transform any Lyee project in a way that it can be put under the control of a NUCM system. In detail, the following steps are necessary to achieve this goal.

First, we concentrate on moving information into the configuration management system. This can be the creation of a new repository when a project is put under configuration management control the first time or it can be the update of the repository contents after changes have been applied to the program. Two steps are to be performed:

- *Transformation.* This initial step scans the LyeeALL database and the additional files used by the LyeeALL tool to store the data of the Lyee project. This also includes additional files as e.g. the dialog screen definition files. After scanning, the transformation groups the gathered information together according to the definitions of the Lyee program structure. Then, the data will be written to a file system that acts as workspace for the NUCM system.
- *Checkin.* During this step the NUCM system performs its built-in checkin operation. So the contents of the workspace are stored in the NUCM repository.

Other steps are required to retrieve the data from the repository. To get a working copy from the SCM repository into the LyeeALL database, a reverse process takes place:

- *Checkout*. First, the data from the repository has to be checked out into a file system. The data stored in this file system will have the well-defined structure based on the definitions of the Lyee program structure.
- *Re-Transformation*. Then, the re-transformation can scan the file system and store the data in the LyeeALL database or create additional files for the LyeeALL tool.

As mentioned at the beginning of this paper, there are two major approaches for software configuration management: the *state-based* approach, realized in the checkin/checkout model, and the *change-based* approach, realized in the change-oriented model. Generally speaking, state based configuration management systems store differences between two versions of a component in a repository, change-based configuration management systems are generalizations of conditional compiling.

One characteristic of Lyee programs is that they do neither consist of multiple source files nor does the LyeeALL tool provide a functionality similar to conditional compiling. This leads to monolithic applications; it is almost impossible e.g. to build different variants of an existing program.

To enhance the flexibility of Lyee programs it is possible to provide a configuration management system offering both, the state-based approach as well as the change-based approach. In [6] it is described how the NUCM system can be used to build a state-based or a change-based configuration management tool.

One aspect of state-based configuration management is that a system can allow to check out one or multiple writeable copies of a software object. This rises the need for a merge operation that ensures consistency when these files are checked back into the system. Using one of the merging approaches mentioned in [14], it is possible to extend the Checkin-step of our proposed system to provide this functionality for Lyee software projects as well.

6 Discussion and Further Work

In section 4 we formulated definitions to represent the structure of Lyee programs. However, these definitions are generalizations of the structure that can be found in Lyee software. They can represent every Lyee program, but not every construct composed using the definitions would be a valid Lyee program (despite the fact that this is impossible due to the ambiguity of the X_ν).

One example: in Lyee programs, the initial pallet of a base structure is always a W04 pallet, thus $\forall d \in D : w_0 \in W_{04}$ with $d_{(3)} = w_0$. Our definition does not necessarily require this. However, the structural restriction of Lyee programs is no problem for our system.

The definitions provide another benefit than formulating a configuration management concept. Based on the definitions it is possible to set up propositions that indicate whether configurations of Lyee software can be made or how a larger project should be split up into parts and how they depend on each other.

An example: assume that we have to process route diagrams p_1 and p_2 in a program. Using the definitions, the defineds 'used' by a process route diagram can be expressed as set. Let d_1 be the set of defineds used in p_1 , d_2 the set of defineds used in p_2 . Now we can suggest

that splitting the program into two configurations e.g. to establish team work will cause no problems if the intersection of the defineds used by the p_1 and the defined used by p_2 is the empty set. In other words: p_1 and p_2 can be developed by different persons, if $d_1 \cap d_2 = \emptyset$. Please note that the intersection will not be empty if p_1 and p_2 refer to e.g. the same database. In this case, splitting can make sense as well, if the data base structure is left untouched.

These considerations show how it is possible to introduce a component-oriented view on Lyee software. In our opinion, this is vital for Lyee methodology. If larger systems are to be developed using Lyee methodology, the possibility of cutting software into handsome parts has to exist. Furthermore, software configuration management only can make sense if there is a component oriented approach in the design paradigm.

The software configuration management concept we described above introduces a second storage of the contents of the LyeeALL database. We chose to duplicate the data in this way as it offers the rather simple possibility to create the software configuration management system as stand-alone part that can be used in combination with the LyeeALL tool. For use in real software production it would be suitable to integrate the software configuration management system into the LyeeALL tool or the LyeeMAX tool.

As mentioned, we can provide a software configuration management system that offers state-based versioning as well as change-based versioning. As we cannot judge at which approach is the best to use as our knowledge about the Lyee software process is still limited, we decided to realize these multiple approaches within the prototype of the system.

The implementation of the prototype of the system is the next part of work that has to be performed. This prototype will cooperate with the LyeeALL tool and be restricted to the use of Visual Basic as target language for the Lyee programs. This makes implementation easier and does not limitate the universality of the validation results.

After building the prototype we will conduct several steps to validate this concept. We will use the building of a partner component for insurances to test the versioning features of our configuration management concept. Additionally, we will introduce variants into this component to especially test the change-based versioning functions.

7 Conclusion

We have redefined the program structure of Lyee software using sets and their relationships. Using these definitions, we were able to formulate a configuration management concept for Lyee software that covers state-based versioning as well as change-based versioning. The evaluation of the system will be performed next to validate the results in this paper. In parallel to the evaluation we will continue to set up rules how larger Lyee projects can be split up to achieve a maximum of independency for the sub-projects. Using this configuration management concept, the software process using Lyee methodology will get another toolkit meeting the needs of large-scale system development.

References

- [1] B. Berliner. CVS II: Parallelizing Software Development. In *Proceedings of the USENIX Winter 1990 Technical Conference*, pages 341–352, Berkeley, CA, 1990. USENIX Association.
- [2] R. Conradi and B. Westfechtel. Configuring Versioned Software Products. In I. Sommerville, editor, *Software Configuration Management: ICSE'96 SCM-6 Workshop*, LNCS 1167, pages 88–109, Berlin, Germany, March 1996. Springer-Verlag.

- [3] R. Conradi and B. Westfechtel. Towards a Uniform Version Model for Software Configuration Management. In R. Conradi, editor, *Software Configuration Management: ICSE'97 SCM-7 Workshop*, LNCS 1235, pages 1–17, Boston, Massachusetts, May 1997. Springer-Verlag.
- [4] S. Dart. Concepts in configuration management systems. In P. H. Feiler, editor, *Proceedings 3rd International Workshop on Software Configuration Management*, pages 1–18, Trondheim, Norway, June 1991.
- [5] P. H. Feiler. Configuration Management Models in Commercial Environments. Technical Report CMU/SEI-91-TR-7, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, March 1991.
- [6] A. van der Hoek, D. Heimbigner, and A. L. Wolf. A Generic, Peer-to-Peer Repository for Distributed Configuration Management. In *International Conference on Software Engineering*, pages 308–317, 1996.
- [7] The Institute of Electrical and Electronics Engineers, New York. *IEEE Guide to Software Configuration Management*, 1988. ANSI/IEEE Standard 1042–1987.
- [8] The Institute of Electrical and Electronics Engineers, New York. *IEEE Guide to Software Configuration Management Plans*, 1990. ANSI/IEEE Standard 828–1990.
- [9] F. Negoro. Principle of Lyee Software. *Proceedings of the International Conference on Information Society in the 21st Century*, 2000.
- [10] F. Negoro. The Predicate Structure to Represent the Intention for Software. In *Proceedings of the International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing*, 2001.
- [11] W. F. Tichy. RCS - A System for Version Control. *Software - Practice and Experience*, 15(7):637–654, July 1985.
- [12] W. F. Tichy. Tools for software configuration management. In J. F. H. Winkler, editor, *Proceedings of the International Workshop on Software Version and Configuration Control*, pages 1–20, Grassau, Germany, 1988. Teubner Verlag.
- [13] B. Westfechtel, B. P. Munch, and R. Conradi. A layered architecture for uniform version management. *IEEE Transactions on Software Engineering*, 27(12):1111–1133, December 2001.
- [14] A. Zeller. *Configuration Management with Version Sets*. PhD thesis, University of Braunschweig, 1997.

An Experiment on Software Development of Hospital Information System

Yutaka FUNYU, Jun SASAKI, Takushi NAKANO, Takayuki YAMANE, Hiroya SUZUKI

Iwate Prefectural University, Faculty of Software & Information Science
Sugo152-52, Takizawa-mura, Iwate-ken, Japan 020-0193
Phone: +81-19-694-2568, Fax: +81-19-694-2569,
E-mail: {funyu, jsasaki}@soft.iwate-pu.ac.jp, taxi@ics.co.jp,
{g231y017, g231a019}@edu.soft.iwate-pu.ac.jp

Abstract. In the medical and welfare fields, there are only a few specialists or experienced people of information systems, which has been making it quite difficult to introduce information systems to those fields. In order to overcome this difficulty, the development of information system by users themselves is required. We understand that the Lyee methodology has a possibility to solve that problem from Lyee's characteristics and have done a basic experiment to introduce the Lyee methodology into medical and welfare fields. This paper shows the results of the experiment on the software development of hospital information system. Then we propose an efficient approach to the software development by using Lyee methodology.

1. Introduction

As the information technologies progress, information systems have been developed in various fields rapidly [1][2][3]. Especially in manufacturing and financing industries, many corporations have apparently successfully introduced them and got great results. On the other hand, however, development of information systems is extremely behind in the area of semi-public sectors such as medical, welfare, education and environment industries [4]. The reason of this, in general, is rather hard to make it effective since the economical rationality is not measured comparing with that of the information systems developed for private companies and the maintenance requires a high degree of skills.

Information systems is developed by a co-working of users and system engineers. To understand well the users requirement, a system engineer who has knowledge and experience of information technologies and software developments is indispensable. However, in the medical and welfare industries, there are no yet advanced information systems. Accordingly we think there are few people who have such experiences. An organization of such industries cannot afford to have group of specialists like systems engineers. Thus, it is considered that in these fields it is much harder for them to develop information systems comparing with other fields, and they have to depend on outside companies to maintain the systems.

To solve this problem, we propose the information system of the users, by the users and for the users. 'Of the users' means that the users themselves can maintain that information systems in through its life cycles. 'Developed by the users' means that users themselves are the main developer of that information system. 'System for the users' means that the users

themselves can obtain the effectiveness of that system. Information system in medical and welfare fields should aim this 'of the users,' 'for the users' and 'by the users.' We strongly believe that the Lyee methodology has the high possibility of being developed and maintained 'by the users' [5].

We considered the characteristics of Lyee methodology as follows;

- With Lyee's basic structure and PRD (process route diagram), it is possible to produce the system automatically, and the development of coding becomes not so necessary.
- Making a PRD is relatively not difficult by extracting words from the requirement specification and defining them.
- Consequently, we obtain higher possibility to develop information system without special knowledge or experiences in contrary with conventional software developments.
- Subsequently, the users will be able to develop the system themselves more easily compared with traditional developments.

According to applying above characterizing, we can develop information system 'of the users,' 'for the users' and 'by the users' in medical and welfare fields. In order to verify this possibility, we have executed the following experiment on software development of hospital information systems.

In section 2, the experiment methods are shown and its results will be shown in section 3. In section 4, we'll analyze and discuss the results and then bring our discussion to the conclusion.

2. Experimental method

There are four experimenters and each experimenter develops an information system respectively by using Lyee methodology. The experimenters consist of two people who have experienced in developing information systems and another two people who do not have experience in development. The system they develop is a part of hospital information system. Since the development scale is relatively small, it is easy to understand the system's image by those people. The experimenters develop the system based on the screen specifications. We record the time to be spent and questions on each development step. We analyze the experimental data, the time and questions, and verify the differential of the experimenters' development approach and the achievement of system development by using Lyee methodology.

2.1 Experimenters

The experimenters consist of two postgraduate students without advanced experience in system namely A and B, and two practitioners namely C and D. One of the practitioners belongs to ICBSMT (The Institute of Computer Based Software Methodology and Technology Inc.). In this experiment, we approach to make their knowledge on Lyee methodology equivalent, three experimenters (except the one who belongs to ICBSMT, named as experimenter D) have attended the Introductory Course of Lyee that is opened by the ICBSMT. Table 1 shows skills and knowledge of each experimenter A, B, C, and D.

Table 1 : Experimenters.

	Experimenter A	Experimenter B	Experimenter C	Experimenter D
Status	Postgraduate student	Postgraduate student	System engineer	System engineer (Lyee)
Experience on Information-system development	×	×	○	○
Knowledge on Lyee methodology	△	△	△	○
Training of “Introductory course of Lyee”	○	○	○	×
Experience of programming(VB)	△	△	○	○

○:well △:middle ×:poor

2.2 Schedule

The term of the experiment is carried for three consecutive days, each day for seven hours, from 9:30 a.m. to 17:30 p.m.. Though each experimenter has taken the Introductory Course of Lyee, this system development requires some techniques which are not include in the Introductory Course. So, all four experimenters including D had supplementary lesson in the afternoon of the previous day. Figure 1 shows the outline of the schedule.

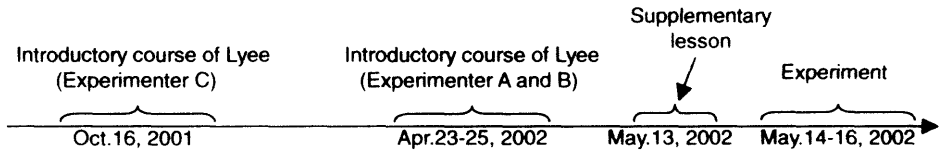


Fig. 1 : Schedule of the experiment.

2.3 Experimental conditions

Using one notebook PC, each experimenter starts the experiment. The experimental conditions are shown on Table 2.

Table 2 : Experimental conditions.

Classification	Items
Development tool	LyeeMAX and LyeeALL2
OS	Microsoft Windows2000
Development language	Microsoft VisualBasic6.0
Database	Microsoft Access2000
Communication tool (for questions and answers)	IP Messenger for Win

2.4 Experimental environment

We prepare a Lyee instructor who answers the experimenters' questions about the Lyee methodology and the system's design. Figure 2 shows the experimental environment. As shown, each experimenter lines up sideways, each equipped with a notebook PC. They are not allowed to speak each other during the experiment, only they can ask questions to the Lyee instructor who has system developing experiences on the Lyee methodology and answers questions impartially to every experimenter.

PCs of each experimenter are connected to a video projector in front of the Lyee instructor through display cable, so the Lyee instructor can keep watching the progress of each experimenter if necessary. The experimenters' PCs and that of the Lyee instructor are connected with LAN, and all the questions and the answers are exchanged with the network communication tool, IP Messenger. By this, each experimenter cannot know the progress of other experimenter's developments or the question contents.

The experimenters have the screen specification of the system but the detail specification of the system or the database of it is not distributed.

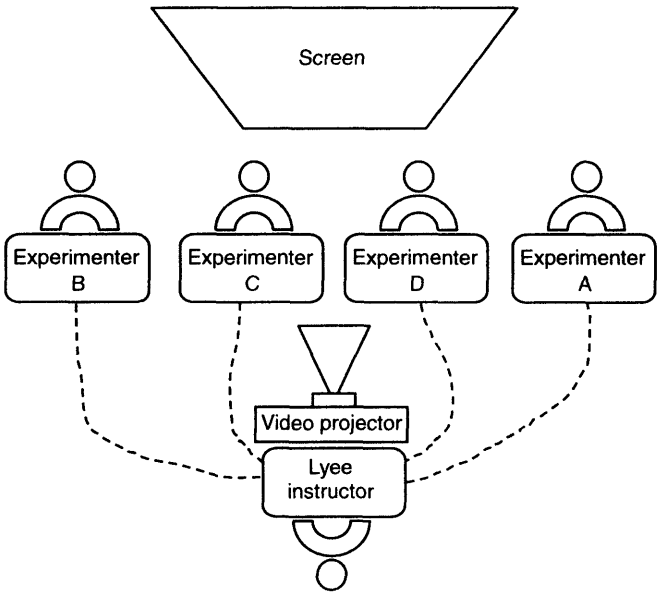


Fig. 2 : Experimental environment.

2.5 Collection of experimental data

In addition to questions, each experimenter is required to make a report when a certain job is completed. By this report, the Lyee instructor will measure his progress in development. Figure 3 shows an example of the conversation between an experimenter and the Lyee instructor.

```

=====
From: Experimenter C
at Thu May 16 09:15:50 2002
-----
I can't understand the way to update records to DB.
I have set 'KA_DBRD.Acceptance_No + 1' on
'Logic Unit ID/Word ID=KA_DBUP/Accept_No'.
=====

From: Lyee instructor
at Thu May 16 09:17:18 2002
-----
Your setting on that word is OK.
The action vector will update records to DB.
=====

From: Experimenter C
at Thu May 16 10:21:48 2002
-----
I must set on 4 words to update records to DB.
But I have set on 3 words, so I couldn't update records to DB.
Now, I can update records to DB.
Next time I will write records to DB.

```

Fig. 3 : Example of conversations between an experimenter and a Lyee instructor.

2.6 An example of a hospital information system

We decided to develop a part of a hospital information system by using Lyee methodology. And we focused the system function to make the analysis easy. The system function is to notify the patients on how long they should wait until they get a medical examination. Because we believe that long waiting-time in a hospital in Japan is one of the significant problems in such system. The goal of the system is to reduce patients' stress by letting them know their waiting-time and to have them use the waiting-time for some other things.

The system functions can be divided into three parts as follows:

- Acceptance; accept a patient's application and issue a reception slip. The system registers the data of the patient who is accepted.
- Notification by a doctor; when a doctor starts to examine a patient, this should be informed to the system. When something interrupts the continuation of doctor's examination, this should also be informed and updated into the overall system. The resumption should also be informed. When doctor's examination is finished, the patient's waiting-time is not added to the general waiting-time of the hospital database.
- Calculation of waiting-time; calculate the patients' waiting-time, and the results should be displayed on a big display in front of examination rooms. The data are to renewed and update every five minutes intervals.

Figure 4 shows the system structure. For this experiment, we limit the function to the acceptance, the waiting-time notification and the calculation of the waiting-time. The experimenters try to develop the system with above three functions using one PC.

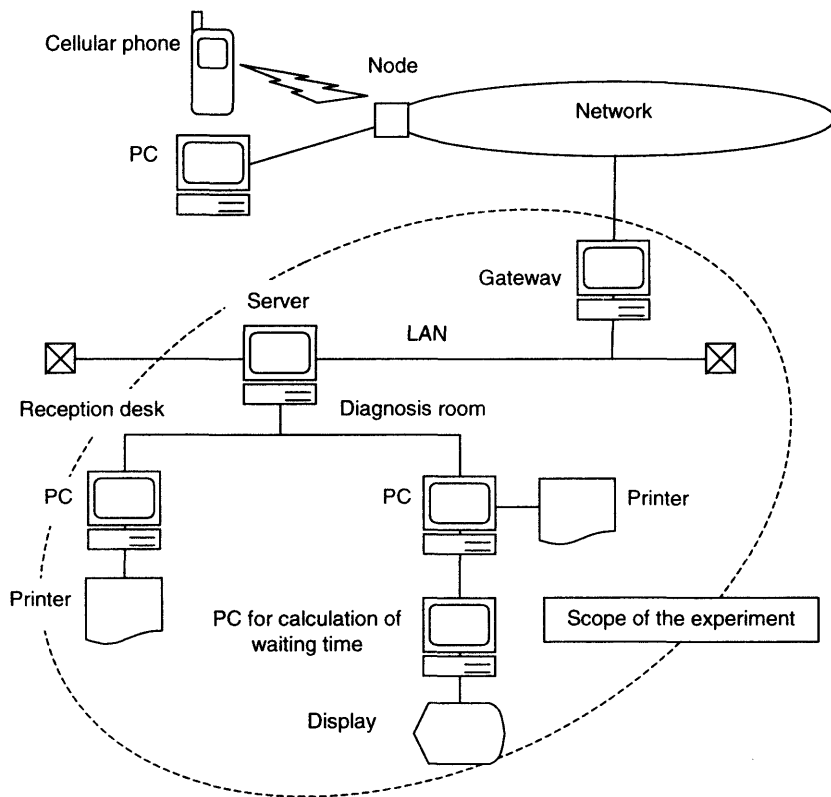


Fig. 4 : System structure of the hospital information system.

3. Results

In this experiment, we have recorded the development processes and time through the conversation log between the experimenters and the Lyee instructor. After the experiment is finished, we collected the experimenters' feedback logs. The following is the result in this experiment.

3.1 Development process and time

We have carried out the experiment to introduce the Lyee methodology into medical and welfare fields. We will show the experimental results as it is to grasp some problems on applying Lyee methodology to the fields.

Though we decided the order of the development process approximately before the experiment, each experimenter took different approaches in the real experiment. But as a result, no experimenter could finish the whole system functions which we had preplanned. These make it difficult to compare simply each time to be spent for the development.

Therefore, we have focused on the relation between two factors, the development process in Lyee methodology and the functions of the aimed system in the experiment. Figure 5 shows the relationship among the two factors and experimental time. Figure 5-(a), 5-(b), 5-(c), and 5-(d) show the results by the experimenter A, B, C, and D, respectively.

The vertical axis shows a development process in Lyee methodology, and the horizontal axis shows the rate of time that each experimenter spent for the development of each. The aimed system has some functions and each function consists of some development processes. Two dots shown at the both ends of the straight time indicate the beginning and the ending of working time by an experimenter. A line length is the rate of each working time in whole working time. We measured the development for each function unit. And we could not measure the further detail time to be spent for each development process. In figure 5-(a), the experimenter A develops some functions from 'a' to 'k' in sequence in the experiment. The function of 'f', for example, is achieved by seven development processes, and it occupies about 25% of the whole experimental time.

The difference of development approach by each experimenter is shown in from figure5-(a) to 5-(d). When we pay attention to the beginning time of LyeeAll2, the experimenter A, B, C, and D, start it at 52%, 43%, 15%, and 55% of the whole the experimental time respectively. We can find the experimenter C start LyeeAll2 earlier than other experimenter does. In figure 5-(d) by the experimenter D, the lines are distributed from the upper left to the lower right. Analysis and discussion of these results will be described in the chapter four.

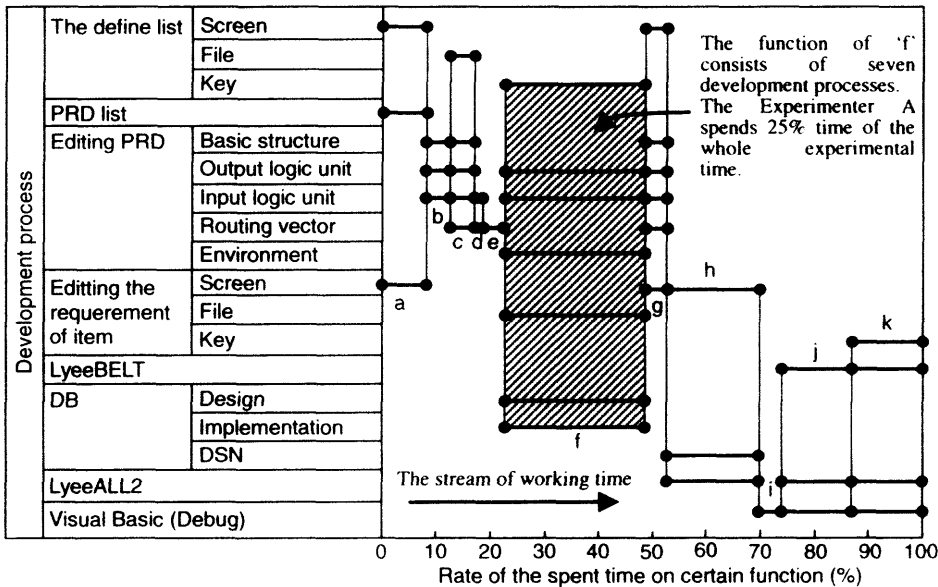


Fig. 5-(a) : Combination between rate of time and development process (Experimenter A).

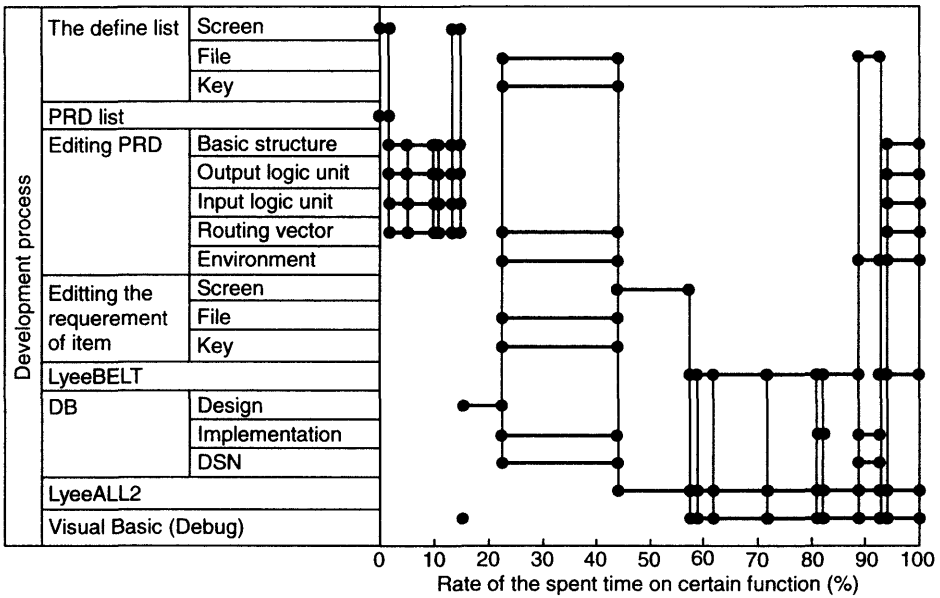


Fig. 5-(b) : Combination between rate of time and development process (Experimenter B).

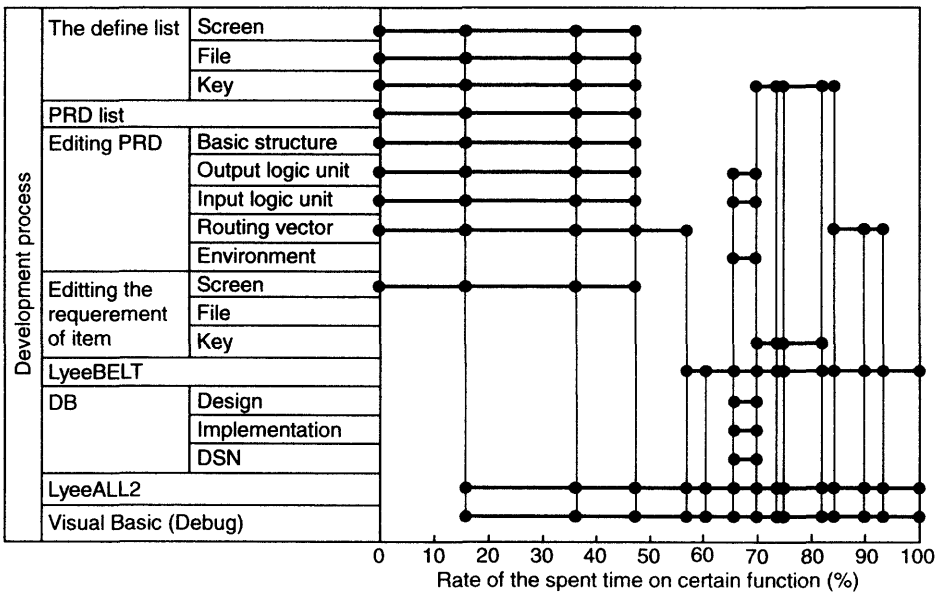


Fig. 5-(c) : Combination between rate of time and development process (Experimenter C).

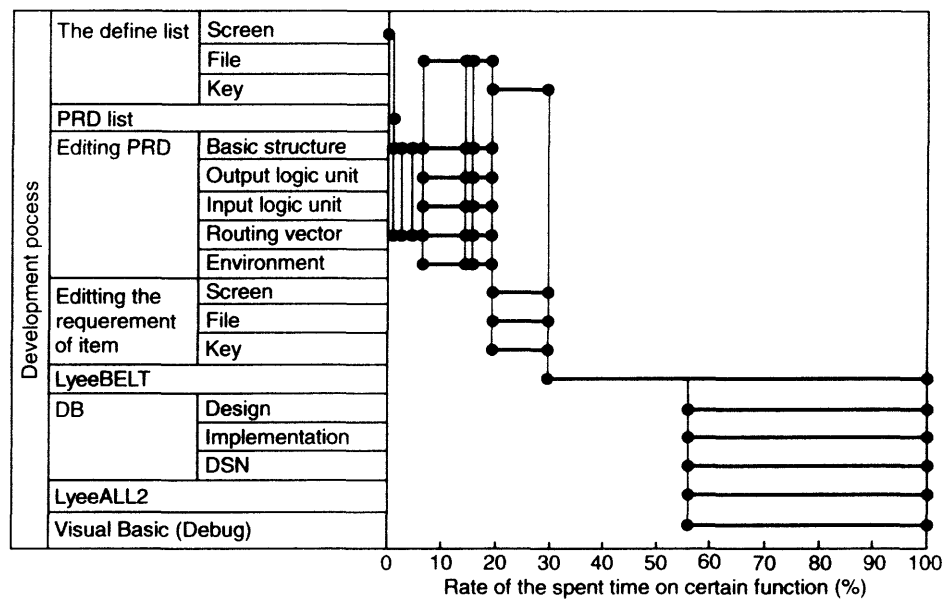


Fig. 5-(d) : Combination between rate of time and development process (Experimenter D).

3.2 Feedback by experimenters

After the experiment, we collected the experimenters' feedback due to the experiment. Table 3 shows their feedback grouped into three types, well evaluated (positive), needs some improvements (negative) and comments. Most of feedback was not related to the principles on Lyee methodology, but on the way of using Lyee methodology in software development. As to the experimenter D, his feedback is not collected since he was a system engineer to be responsible for Lyee software development.

Table 3 : Feedback by experimenters

Experimenter		A	B	C
Process of Lyee	Registering the defined			
	Editing PRD		--	+++!
	Editing the requirement of item			
	LyeeBELT		---	-!
	LyeeALL2	-	---	+
	Debug (Visual Basic)		-	--
	Others	--!!!!		++--!!!
Usage of Lyee	Method to deal with Database	-		
	Specification			+!
	Method to learn Lyee	!!		--!
	Revise after running program	-		!
	Comparison with old version			+
	Experimental method	!!		

+ : positive - : negative ! : comment

4. Analysis and discussion

4.1 Development process and achievement level

Figure 6 shows the development processes and the achievements level of each experimenter. The experimenter C had a development approach limiting all his efforts to one PRD and had succeeded in developing the PRD. We call this developing method as ‘systems function approach’ (SFA). While the experimenter D developed all the PRDs at the same time, however he could not complete all the developments. We call his method as ‘development processes approach’ (DPA). The experimenters A and B took their approaches in between these two.

When a developer takes the SFA, he can make the function to be clear. And he could complete the software, by dealing with only one PRD. The work for confirmation of users’ requirements and understanding the DB design, can be earlier than that by the DPA. We also consider SFA has quicker speed of learning about Lyee methodology than DPA does from conversations in the experiment.

As a result, we think SFA is effective in the development by utilizing the characteristics of Lyee methodology.

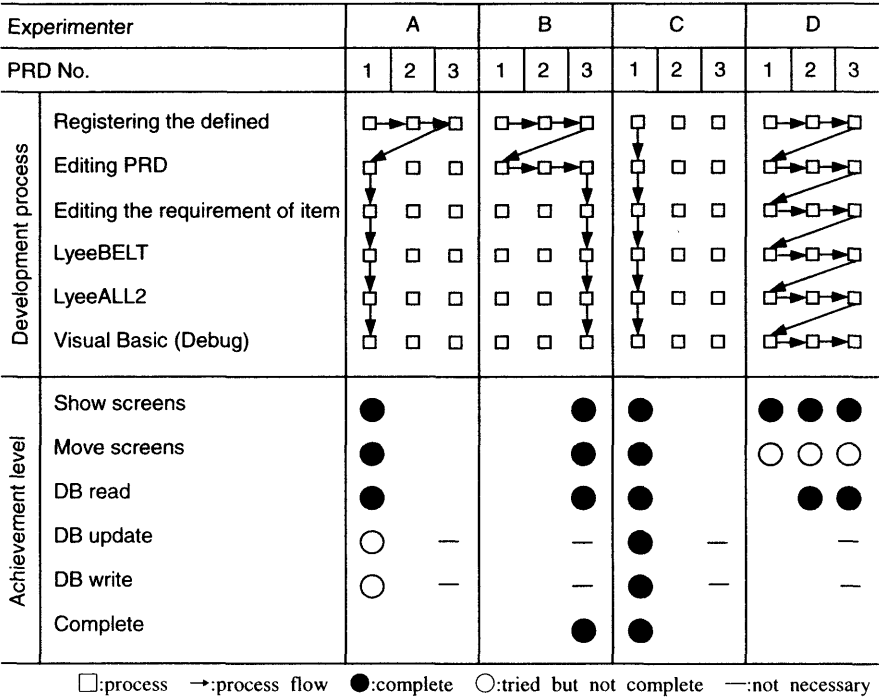


Fig. 6 : Development process and achievement level of each experimenter.

4.2 Interference problem of database access

Lyee methodology prepares the basic structure of files to access to the DB. Each basic structure is connected by routing vector in the PRD.

A basic structure of files, as the other basic structures, is defined in the PRD, and basic structure never interfere with each other. Though the basic structures of files are different, we may deal with the same DB file. Figure 7 shows such a case. In this case, if we design DB without data consistency the interference problem of DB access occurs. We think it is necessary to complete the design of the DB in early stages of the development process, especially if it is carried out by a group of developers.

In this experiment, each experimenter had difficulties in the DB design and the way of access to the DB. Especially, the experimenter A and B spent much longer time in the works related to the DB definition. Of the total development time, the experimenter A took 32% on this works, and the experimenter B for the same work took 39%. On the other hand, the experimenter C took 19%, much lower than the others for the same workload. As the experimenter C made the specifications, such as designing of the screens used in the experiment, he had better understandings of the DB definitions than the others.

Lyee methodology has characteristics both users and developers have a common recognition through the preparation of PRDs. In order to utilize this characteristics, we think DB design have to be done in the early stage of the development. We consider that the DB design in Lyee methodology is as important as in other development methods.

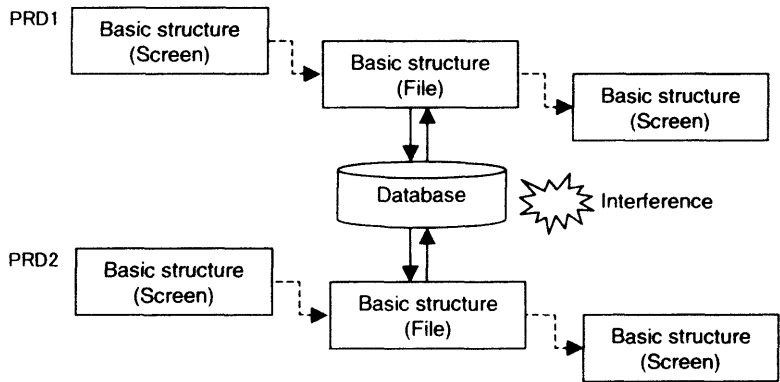


Fig. 7 : Interference problem of database access.

4.3 The number of questions and reports

We classified conversations between experimenters and the Lyee instructor into two groups, questions and reports. Figure 8 shows change of questions and reports. In the horizontal line, the time is shown in six sections, each representing the morning time or the afternoon time for the three days of the in the experiment.

From these figures, we know that the experimenter C asked questions and reported the completions of each step with the same pace through the experiment, on the other hand the other experimenters had fewer conversations with the Lyee instructor. We understand from their comments after the experiment that no question means not enough understanding, this is due to low understanding on Lyee methodology. The reason why the experimenter C

could ask many questions and report the completions is related with the fact that he took the SFA as we stated in 4.1. He can make the function to be clear, so he can question about the problem faces him.

The number of question by all experimenters was 71. Only 3 questions were about system specifications of the aimed system, and the rest referred to on how to use the Lyee methodology. This fact shows much time spent by each experimenter was used for learning Lyee methodology. We consider that they cannot easily understand how to use Lyee methodology, because they didn't have available manuals for Lyee methodology, and some terms in Lyee methodology are difficult. Those are problem to be solved in the near future[5].

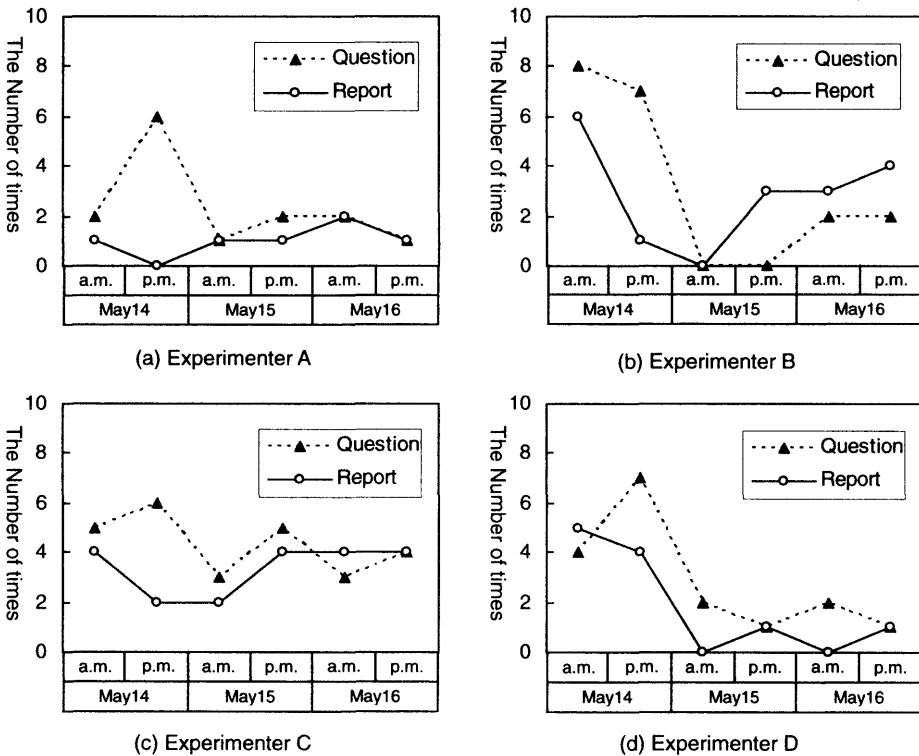


Fig. 8 : The number of times of each experimenter.

5. Conclusions

This paper shows the results of the experiment on the software development of hospital information system. Then we propose an efficient approach to the software development by using Lyee methodology. There are four experimenters and each experimenter develops an information system by using Lyee methodology.

- In the case of clearly defined specifications, it might be effective to develop multiple PRDs simultaneously. But, in the case of not clearly defined specifications,

it is more effective in developing the system to completing one PRD first, and then to moving to another PRD, and so on. We call this developing method as 'systems function approach' (SFA) , and propose SFA as a development approach using Lyee methodology.

- In Lyee methodology, each PRD is independent and never interfere each other. But, it has problems, like other conventional methods, of accessing to the same DB file through different PRD.
- The experimenters of this experiment spent so much time for learning about Lyee methodology. This came from the facts that there is no available manual and that some of Lyee terms used are hard to understand.

We point out the problems about DB and about learning of Lyee methodology, and we propose SFA to realize an efficient development approach. If we can solve above three problems, there will be high possibility to apply Lyee methodology to medical and welfare fields. Especially, we would like to use more easily Lyee methodology, and we are planning on developing a domain model for Lyee methodology in the new stage.

References

- [1] "2001 White paper information and communications in Japan". Ministry of public management, <http://www.soumu.go.jp/hakusyo/tsushin/h13/index.htm>
- [2] "2002 White paper information and communications in Japan". Ministry of public management, <http://www.johotsusintokei.soumu.go.jp/whitepaper/ja/h14/index.html>
- [3] Kazuko Katase: Trend and View of the Information Infrastructure and the Application in Enterprises, Technical report of IEICE. OFC, Vol. 97 Num. 412 pp.37-43 (1997.11)
- [4] Hitoshi Kuze: Aiminy at the activation of Social Educational Facilities by Informalization. Kyoiku Jyoho Kenkyu, Vol. 14 Num. 3 pp.61-66 (1998.12)
- [5] Lyee-world, ISBN4-901195-06-9
- [6] Katsumi Yasuda and Hamid Fujita : Study on algorithm to realize automatic conversion between programs of different structures, <http://www.ssgrr.it/en/ssgrr2002w/papers/39.pdf>
- [7] Kenji Hiranabe. XP(Extreme Programming): Software development process nouvelle vague part I: Overview of XP and its surroundings, IPSJ Magazine Vol.43 No.03, pp235-241, 2002
- [8] Kenji Hiranebe, XP (Extreme Programming): Software development process nouvelle vague part II: An XP practice report, IPSJ Magazine Vol.43 No.04, pp427-434, 2002

Chapter 8

Automatic Software Generation and Requirement Verification

This page intentionally left blank

Describing Requirements in Lyee and in Conventional Methods: Towards a Comparison

Gregor v. BOCHMANN

School of Information Technology and Engineering, University of Ottawa, Canada

Abstract: The Lyee software development methodology has been recently introduced and suggests that the software development process can be largely simplified by starting with the capture of the user requirements and then generating the implementation code largely automatically, skipping in a sense the software design stage. This paper makes an attempt at comparing the notational concepts that underlie the Lyee methodology with the concepts that appear to be important for the description of software system requirements as defined by the main-stream software engineering methods. While we identify several relations between concepts in Lyee and corresponding main-stream concepts, we also point out a number of important differences, and we leave the reader with a number of questions which, we think, merit further discussion.

1. Introduction

It is well known that the development of larger software systems is a difficult and costly process. Many software development methodologies have been proposed to improve the efficiency of the process and the quality of the generated implementation code. It seems that one should not look for a “silver bullet” that provides a magic solution, but rather one has to look at many aspects of software development, including the choice of notations for describing system requirements, software architectures and designs, and the choice of the development process and the tools used during that process.

In this context, the Lyee methodology has been described in recent papers [Nego 01a, Nego 01b] and has been used for a number of commercial software developments. This methodology proposes to largely skip the development step of creating the software design by providing a tool, called LyeeAll [LyeeAll], which leads from the description of the system requirements more or less automatically to the generation of executable code. The nature of the requirements description supported by this tool is quite different from the familiar concepts used in main-stream software engineering.

The purpose of this paper is to situate the Lyee methodology in respect to the main-stream software engineering approaches, concentrating on the notations used for describing system requirements. We focus on the concepts of requirement descriptions because this is the stage that is emphasized by the Lyee methodology, and it is also the stage from which the main-stream methodologies start. Since the area of requirements is still a field of active research in the software engineering community, we begin the discussion in this paper, in Section 2, with a personal review of what the important main-stream concepts for requirement specification are. In Section 3, we first explain that this paper is based on our limited knowledge of Lyee which is obtained from presentations and papers on the Lyee methodology and reading examples of system specifications. The most pertinent references

on the Lyee methodology are given. We start the comparison by commenting on the main concepts of the Lyee methodology as seen from the conceptual viewpoint of the main-stream software methodologies. Then, in Section 3.3, we present the main concepts of Lyee in more detail and relate them to main-stream software engineering concepts. Some important differences between Lyee and main-stream methodologies are pointed out in Section 3.4. Section 4 contains a discussion of certain questions that arise from this comparison, and is followed by the conclusions.

The discussions in Sections 3 and 4 refer at many places to the “Room Booking” example presented in [Sali 01]. We give an overview of this example in the Annex, and also present an alternate description of its requirements based on main-stream software engineering concepts.

We hope that the discussion and comparison presented in this paper, which represents the personal opinion of the author, could be useful in the further development of the Lyee methodology and its integration with other software development methods and tools.

2. Basic concepts for describing requirements and programs

The notations used to describe computer programs and the requirements that such programs should satisfy have developed over the last 5 decades from relatively low level notations to more abstract concepts, corresponding to the concern which concentrated first on how to write programs and later, when these programs became more complex, on how to specify the properties that the programs to be developed should satisfy. The purpose of this section is to give a (personal) overview of the most important concepts that have been used, and are still being used, when programs and their requirements are defined. Since these concepts have been found useful over the years by many practitioners, it is believed that they represent a good foundation to which the concepts of Lyee could be compared.

If we skip machine programming languages, since they are not oriented towards human understanding but ease of execution by the computer, we should first consider the concepts of high-level programming languages. A well-known book by N. Wirth was entitled “Algorithms + data structures = programs”. This indicates that there are two aspects of programs: (1) the algorithmic aspect, also called control flow, which indicates the possible order of execution of certain basic program actions and (2) the data structure aspect which defines the data structures that are processed by the basic actions mentioned in the algorithmic part.

The main concepts for describing control flow are the following:

- Sequential execution.
- If and Case statements indicating under which conditions which subsequent actions should be executed.
- Goto statement indicating that the next action is out of sequence; this concepts is considered harmful for writing easily understood specifications, and the principles of Structured Programming indicate how one can write programs without using this construct. However, equivalent concepts also appear in higher-level design and requirement notations, such as the flow-chart-like notations for state transition diagrams and Petri nets (and the derived notations for the UML Activity Diagrams [UML] and Use Case Maps [Buhr 98]).
- Structured loops (While, Repeat until, Loop forever).
- The procedure call mechanism including the passage of input and output parameters. This concept is very important, since it represents the main tool for defining procedural (i.e. control flow) abstraction. An interesting feature, in this context, is

recursion (not supported in all cases), which is important for processing recursive data structures, such as lists and trees.

The main concepts for describing data structures are the following:

- Basic data types, such as Integer, Boolean, Character, Bitstring or Octetstring.
- Data type constructors, such as Record (also called Struct), Discriminate Union, linear list (also called SequenceOf, or Array); including the recursive use of Record definitions allowing the definition of arbitrary graph structures, e.g. trees.
- Database schema constructors, including the above Record, but also the concept of a set of records (called Relation) and the concept of relationship between record types.

A third category of concepts is related to the architectural structure of software and hardware systems. They become essential for describing large systems consisting of many smaller components. The main concepts are the following:

- The system (to be constructed) and its environment.
- System components and their sub-components.
- Interfaces between components (and similarly between the system and its environment).

Historically, there were a number of important trends to make the development of computer programs more manageable. These trends went along with additional concepts of which the following appear to be the most important ones:

- Development of high-level programming languages (with the introduction of most of the concepts above).
- Structured programming with the aim to make the control structure of programs more easily understandable. One aspect was the avoidance of the Goto, another aspect was the relationship between data structures (introduced in Simula67 and Pascal) and corresponding control structures (which is another idea reflected in the book title “Algorithms + data structures = programs”).
- Entity-relationship modeling of the data that is important for database applications [Chen 76].
- Data flow modeling (including its formalization as attributed (or colored) Petri nets).
- Information hiding, component interfaces, and object-orientation are aimed at a methodology for subdividing a system into components and sub-components. This goes hand in hand with defining the interfaces that a component provides for communication with the other components in the system. In the object-oriented paradigm, a component is called an “object” and the concept of “data type” is extended to the concept of “object class”; the interface of an object is only defined as far as its signature is concerned, but not concerning dynamic properties required for the output parameters.
- Pre-and Postconditions, specified in first-order logic (independently proposed in two papers by Floyd and Hoare in 1969) and included as debugging aid in the Eiffel language), allow the definition of the dynamic properties of procedure and function calls, as well as the dynamic properties of object interfaces.
- The concept of concurrency is important for operating systems and multi-user applications. It implies the consideration of interprocess communication and mutual exclusion for the access of shared resources. In the context of databases, it has led to the concept of a transaction, which is fail-safe and will be executed either completely (without interference of other concurrent transactions) or not at all.

Many of these concepts have been included in modern notations, for instance in the programming language Java and in the Unified Modeling Language (UML) [UML]. In the

context of this paper, we are mainly interested in the description of system requirements. UML includes a number of different notations intended for the description of system requirements, designs, and implementation structures. The UML Class Diagram provides a notation with a semantics very similar to traditional entity-relationship modeling. The Sequence and Collaboration Diagrams of UML are similar in nature to the Message Sequence Charts defined by the ITU [Z.120]. Also the UML State Diagrams have their correspondence in the SDL language [SDL]. All these notations are best suited for the description of system designs when one defines the behavior of the system components and their interworking. For describing system requirements, UML proposes the Use Case Diagrams and the Activity Diagrams. The former is very rudimentary; it defines the users and the use cases they may be involved in, but the properties of these use cases are simply explained in English. One may use an Activity Diagrams to describe the properties of a use case in a more formal manner. This notation has a semantics quite similar to Use Case Maps [Buhr 98], although the graphical notation is quite different. We think that the notation of Activity Diagrams or Use Case Maps are quite suitable for describing system requirements, including the dynamic properties. A related notation is also proposed in [Boch 00d]. Therefore we compare the Lyee methodology with the use of Activity Diagrams by considering a small example in the Annex. We note that we concentrate in the following on the semantic meaning of the concepts used in Lyee and Activity Diagrams, because we think that the graphic or textual representation of these concepts is of secondary nature.

3. Comparison of Lyee with conventional methods for describing requirements

3.1 What is the Lyee methodology ?

Before attempting a comparison of Lyee with conventional methods, one should have a good understanding of both, the Lyee methodology and the conventional methods. Section 2 tries to give an overview of the conventional methods. Concerning Lyee, I must say that I found it not easy to grasp the meaning of the Lyee concepts and notations. Among the various papers available in English, the following were most useful to my understanding. I found [Nego 01a] useful for the definition of the names of concepts used in Lyee. I found the best explanation of the meaning of the Signification Vector associated with the Pallets in [Nego 01b]. [Poli 02] talks about the same things in an easily understandable language, but remains relatively vague. I was therefore looking for a more concrete (operational) definition of the Lyee concepts, and some examples. I think that the case study of [Salinesi 01] is the most interesting example for this purpose. Note, however, that the concepts of Logical Unit and Word are called Defined and Item, respectively, which corresponds to names introduced in [Roll 01].

The paper [Roll 01] is interesting because it tries to define a meta-model for the concepts of the Lyee methodology (called concepts of the "Lyee software requirement layer"), and at the same time, the paper proposes a simpler meta-model (called "User requirements layer") which corresponds to a more abstract view of the Lyee concepts. In the subsequent comparison, we will refer to both of these layers. These two meta-models are shown in Figure 1.

It seems that the main argument for the promotion of the Lyee methodology is the statement that, using this methodology, the developer can easily define the requirements of the application and obtain the corresponding program largely automatically using the Lyee development tool [Lyee-All]. This means that the traditional software activities related to the subsequent development of the software design and implementation (including the related testing activities) would be minimized. Since the emphasis is put on the definition

of the requirements (and this definition must still be made by the software expert in conjunction with the user), we concentrate in our comparison on aspects related to the definition of the system requirements.

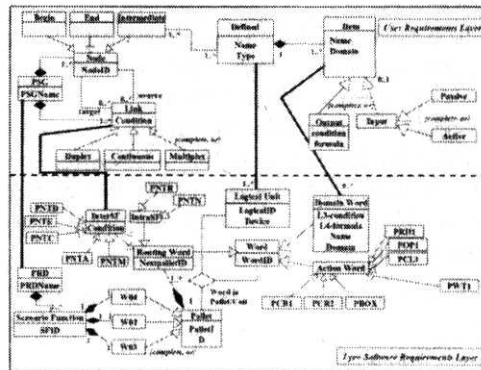


Figure 1 (from [Roll 01]): Meta-model of Lyee concepts and a more abstract view

Based on my current understanding of the Lyee methodology (which is certainly incomplete), I came up with the following comparison between the Lyee methodology and the traditional approach to requirements description. I hope that this discussion and comparison could be useful in the further development of the Lyee methodology and its integration with other software development methods and tools.

3.2. The Lyee methodology seen from the main-stream conceptual viewpoint

In Section 2, we identified three main categories of concepts for describing information processing systems, namely concepts for describing (a) control flow, (b) data structures and object classes and relationships, and (c) the architectural structure of software and hardware systems. The third category of concepts is of minor importance for the description of requirements, since requirements, by definition, describe the properties and behavior of a system as a whole, in terms of its interactions of the environment. The architectural structure of that system should normally be elaborated during the system design process, and not given as part of the requirements. Therefore we concentrate our attention in the following on the concepts used for describing (a) control flow and (b) data structures, objects and relationships.

It is in the area of control flow that much of the originality of Lyee lies. In fact, one may identify two three levels of control flow within the Lyee methodology. At the upper layer, there is the high-level control flow, expressed by the Process Route Diagrams (PRD). At this level, application-specific control flows are specified; this level has in fact much similarities with the main-stream activity diagrams, as shown in the example of the Annex (see also subsequent discussion in Section 3.3). At the next layer is the control flow within each Scenario Function. Here Lyee defines a standard control flow in the form of a loop (as seen for example in Figure 9 for SF1 and SF2, etc.). At the lowest level is the control flow within each of the three pallets of each Scenario Function. Here again, Lyee defines a standard control flow (as for instance depicted in Figure 3 of [Nego 01b]).

A major claim of the Lyee methodology appears to be the claim that this standard control flow at these lower two levels results in an important simplification of the requirements capture. This appears to be the major advantage of the Lyee methodology. It would be interesting to perform some controlled experiments in order to validate this claim.

In the area of data structures, objects and relationships, one can identify within the Lyee methodology again several “levels”. At a higher level, we have application-specific data structures, called Logical Units, which have some similarities with object classes with attributes and method interfaces (as discussed in Section 3.3 in more detail). At a lower level, Lyee has on the one hand standard data type primitives, such as Integer, String, etc., like most programming and specification languages. On the other hand, Lyee includes some standard and rather complex data structures to support the processing of the “Words” (attributes) of Logical Units in relation with the standard control flow described above. Fortunately, in many cases, the latter data structure need not be visible to the person defining the requirements. It is my personal impression that more powerful concepts for describing object classes and relationships at the higher level would be useful for the Lyee methodology, as discussed in Section 3.4.

3.3. Correspondence between concepts

In the following, we list some Lyee concepts and discuss their meaning as compared with concepts familiar from the conventional methods for requirements description.

Domain Word: A Domain Word has an identifier, a name and a domain, which is a basic type (see last column in Figure 8). It is not clear what the difference between the identifier and the name is, since most examples use the same string for these two purposes. They represent input/output parameters related to a Logical Unit, and they also correspond to variables associated with the Pallets of Scenario Functions.

Logical Unit: A Logical Unit represents an interaction with the environment of the Lyee application. It has much similarity with an Remote Procedure Call (RPC) performed by the application accessing a component in the environment which is identified by the Type of the Logical Unit. For instance, as shown in Figure 10, the *CostomerDB* Logical Unit represents a query of the database getting the name of a customer given his identifier. Also a Logical Unit of Type *Screen* can be considered an RPC on the screen object which provides as output Words from the application (which become input parameters for the screen) the information to be displayed on the screen, and the response from the user is provided as input to the application.

If we consider that the components in the environment of the Lyee application are objects, then the Logical Units associated with a particular object can be considered to be defined methods on these objects. The output Words associated with a Logical Unit correspond to the input parameters of that method, and the output parameters of the method (we assume that there may be several, in contrast to popular languages such as Java) correspond to the input Words of the Logical Unit.

Scenario Function: As shown by the example in Figure 9, a Scenario Function contains three so-called Pallets, one for defining output (W04), one for receiving input (W02) and one for determining whether and where to continue the processing (W03). The semantics of the Scenario Function seems to be the very essence of Lyee. In simple situations, as for the example shown in Figure 9, the semantics corresponds to one or several method calls on objects in the environment and a subsequent choice of the next Scenario Function to be executed. As shown in the Lyee meta-model of Figure 1, the description of the possible next Scenario Functions and the conditions that are associated with this choice are contained in the Routing Words and the InterSF.Condition.

Process Route Diagram (PRD): While the low-level control structure of the Scenario Functions is a standard structure in Lyee, there are many application-dependent features which are defined using a number of tables provided by the user interface of the LyeeAll tool. The Process Route Diagram is a convenient overview of many important properties of the Scenario Functions that are included in an application. As exemplified by Figure 9, it shows the overall control flow of the application and may include information about the interactions with the environment and informal comments about the conditions for the control flow choices. This information corresponds largely to the information provided in a UML Activity Diagram, as shown in Figure 7, for example.

3.4. Differences

Besides the corresponding concepts discussed above, there are a number of important differences, such as the following.

Flat aggregation hierarchy: The Words included in a Logical Unit as input or output parameter are of basic type, it is not possible to consider Words that are objects with attributes, for instance. In the object-oriented approach, an attribute of an object may be another object, and so on. This is very useful for describing complex aggregation relationships. This is not possible in Lyee.

No modeling of relationships: Lyee provides no tools for modeling relationships between object classes, as used in entity-relationship modeling and UML Class Diagrams.

Abstraction: In Lyee, there seems to be no way of introducing procedural abstraction; there is no procedure call mechanism with parameter passing.

Relying on the underlying implementation language: For describing conditions for the choice of control flow or for the validity of input received, Lyee has no own notation, but relies on the notation of the programming language into which the Lyee specification is translated when an implementation is generated. The same holds for the description of the expressions that define output values.

Arrays: The MBA Database case study [MBA 01] includes the use of an Array concept in Lyee. However, it is not clear how this can be used. It seems that for each Domain Word that is declared to be an array, there must be another Domain Word that indicates the length of the array. Different Domain Words of type array may form the rows of a table in the user interface (where each array index corresponds to a table row). Data processing seems to be defined only within each row (without interference between different array index values; that is, $A[I] := B[I+1]$ is not allowed).

4. Discussion of further issues and questions

Although there exist some correspondence between certain concepts of Lyee and the traditional concepts of programming languages and requirement descriptions, it is clear that the Lyee methodology for software development is quite different from current mainstream approaches. Some further issues and questions are identified in the following.

Lyee's application domain: Most applications of Lyee that I have seen involve a database and users performing read and/or update transactions on the database. An important aspect is the graphical user interface, the validation of input data and the presentation of the appropriate results. It is not clear how much the Lyee methodology is suitable for other kinds of application domains. The following features of Lyee may limit its usability in certain areas: (1) There are no recursive procedures which makes the description of the processing of complex data structures, such as trees, very difficult. (2) There is not much support for the decomposition of a system into separate components (well, you may notice that this is a software design issue, and not related to requirements); this makes the construction of distributed systems with Lyee more difficult. Therefore we have the following question: What is the intended domain of application of the Lyee methodology?

Boundary software: Once the requirements of a system to be developed have been determined and entered into the LyeeAll tool, this tool will automatically generate implementation code for the application. However, this generated code requires so-called boundary software, which is written by hand, for realizing the interactions with the components in the environment (such as screen, database, etc.). In some sense, this includes also the software for creating the graphical user interface (for which other mainstream software tools are used). The question then is: How important is the effort to create the boundary software? – Again, it seems that the case of distributed applications has a disadvantage, since it involves communication between several components, of which one or several may be created using the Lyee methodology.

The declarative nature of Lyee: One of the stated advantages of the Lyee methodology is the fact that the lower details of the control flow, that is, the one within each Scenario Function, has a standard form (see for instance Figure 3 in [Nego01b]). This makes the understanding of the programs easier. Through the recursive loop in each Scenario Function, and similar embedded loops within each Pallet, the nature of a Scenario Function appears to be like a mathematical function which determines its output from the state of variables set by the execution of previous Scenario Functions and the own input received. It also includes the checking of input validity and processing of exceptional situations (which, however, is not automatic, but must be specified by the designer, unless the default processing is desired). Let us consider some specific examples:

- **Pallet recursion:** Let us assume that the values of three output Words B, C and D depend on the value of an input Word A, and are defined by the following assignment statements (which are part of the W04 Pallet and must be written in the associated implementation language) “ B := A; C := D + 1; D := B * 2; “. Since during the first execution of the W04 Pallet, the value of D is not defined when it is required for the calculation of the value of C, the latter value will remain undefined and the Pallet is executed a second time during which the yet undefined values will be determined (if possible).
- **Circular definition:** In the case of circular definition, e.g. “ B := A; C := D + 1; D := C * 2; “ the Words involved in the circularity will remain undefined. Question: For what is Pallet recursion useful?
- **Programming loops:** It is possible to use the inherent recursive repetition of a Scenario Function (for instance in Figure 9, the loop from the exit of Pallet W03 in SF01 back to W04, the beginning of this Scenario Function). For example, a loop that could be written in a programming language as “A := input; loop while A < 1000 { A := A * 2; } “ could be realized in Lyee notation [Arai 02] by two Scenario Functions, one reading the initial value of A and the other looping until the termination condition is satisfied (this checking would be done by the W03

Pallet). Question: Is this a natural example ? - It seems to me that the Lyee methodology was not developed for writing such examples. But what should one do in cases where the requirements say: "Find the largest power of A that is smaller than 1000" ?

Meaningful identifiers: We teach our students to use meaningful identifiers in programs and requirements definitions. All Lyee system specifications use numeric identifiers for Logical Units (e.g. screen1, screen2, etc.), Scenario Functions (e.g. SF1, SF2, etc.) and for most other things in Lyee specifications. The only exception are the Word identifiers that are often meaningful. Question: Are these numerical identifiers just bad practice, or does the LyeeAll tool limit the form of identifiers that can be used, or is there a good reason for using numerical identifiers ?

Complex conditions or data manipulations: In the example of the Room Booking example in the Annex, the definition of the database queries (in terms of an SQL query) was left to the boundary software, which means that it was realized outside the Lyee framework. Question: How could complex conditions or complex data processing algorithms be described within the Lyee framework ? - For instance, could one use the Lyee methodology to define a program fragment that realizes the following user query in the context of the MBA Database Application System [MBA 01] which includes the course offerings and the course schedules of students: "Please, display all courses that have no conflict with my current course schedule".

5. Conclusions

In conclusion, we can say that the Lyee methodology is an intriguing approach to software development and it is difficult to understand from the perspective of the current main-stream software engineering paradigms. This paper makes an attempt at comparing the concepts that underlie the Lyee methodology with the concepts that appear to be important for the description of software system requirements within the main-stream software engineering methods. While we identified several relations between concepts in Lyee and corresponding main-stream concepts, we also pointed out a number of important differences, and we leave the reader with a number of questions which, we think, merit further discussion.

Acknowledgements

I would like to thank Fumio Negoro and his team for many interesting discussions and my initial introduction to the Lyee methodology. I also would like to thank Hamid Fujita for encouraging me to work on this topic and for many fruitful discussions. Finally, I would like to thank Carine Souveyet (Paris) and Osamu Arai (Catena, Tokyo) for answering some of my more detailed questions.

References

- [Arai 02] O. Arai, personal communication, June 2002.
- [Boch 00d] G. v. Bochmann, Activity Nets: A UML profile for modeling work flow architectures, Technical Report, University of Ottawa, Oct. 2000.
- [Buhr 98] R.J.A.Buhr, Use Case Maps as architectural entities for complex systems, IEEE Tr. SE, Vol. 24 (12), pp. 1131-1155, 1998.

- [Chen 76] P. P. Chen, The Entity-Relationship model - Toward a unified view of data, ACM Trans. on Database Systems, Vol. 1, No. 1, March 1976, pp.9-36.
- [LyeeAll] Software development tool built by Catena Corp. and the Institute of Computer Based Software Methodology and Technology.
- [Nego 01a] F. Negoro, Intent operationalisation for source code generation, in Proc. SCI 2001, Orlando, Florida, USA, July 2001.
- [Nego 01b] F. Negoro, A proposal for requirement engineering, in Proc. ADBIS-2001, Sept. 2001, Vilnius, Lithuania.
- [MBA 01] The Institute of Computer Based Software Methodology and Technology, MBA database application system by using Lyee methodology, Internal Report, April 2001.
- [Poli 02] R. Poli, Automatic generation of programs: An overview of Lyee methodology, Draft report, university of Trento (Italy) and Mitteleuropa Foundation, March 2002.
- [Roll 01] C. Rolland and M. Ben Ayed, Meta-modelling the Lyee software requirements, in Proc. of an intern. conference, pp. 95 – 103.
- [Sali 01] C. Salinesi, M. Ben Ayed, S. Nurcan, Development using Lyee: A case study with LyeeAll, Internal Technical Report TR1-2, University of Paris I and ICBSMT, Oct. 2001.
- [SDL] ITU-T, Rec. Z.100: System Description Language (SDL), Geneva, 1999.
- [UML] OMG, Unified Modeling Language Specification, Version 1.4, Mai 2001, <http://www.omg.org>
- [Z.120] ITU-T, Rec. Z.120: Message Sequence Chart (MSC), Geneva, 1999.

Annex: The Room Booking Case Study

The Room Booking Case Study is described in [Sali 01]. It describes the use of the Lyee methodology for the development of a program which provides a service for booking hotel rooms. The program has a graphical user interface. In the following, we give an overview of the requirements for this room booking system based on the documentation in [Sali 01], and provide an Activity Diagram and related Class Diagrams which describe the same requirements in UML.

A.1. Room Booking requirements described in UML

A.1.1. Informal description of the system requirements

The following is a summary of [Sali 01], Section 1. The Room Booking application uses a database that contains information about hotel rooms available in different cities. It also interfaces with the user and lets a user select a room for a certain period and make the reservation. A sketch of the user interface is provided in Figures 2 through 4. Figure 2 shows the initial screen where all the defined fields are input by the user. After pushing the “OK” button, the user sees either the screen of Figure 3 (where the *Message* may either read “The customer xxx does not exist in the database” or “No room is available for yyy” and xxx is the customer identifier and yyy is the customer name) or the screen of Figure 4, where all fields are output by the system. The user may then confirm the reservation by pushing the “Validate” button, or cancel the reservation and come back to the initial screen. The schema of the database that is used is given in Figure 5.

Customer ID

Begin Date

End Date

Hotel Category (stars)

City

OK Quit

Message

Return Cancel

Hotel Name

Period : Begin

End

City

Room Number

Stars

Validate Cancel

Figures 2, 3 and 4 (from [Sali 01]): (1) Initial screen, (2) message window, (3) validation

Hotel (HotelID, HotelName, Stars, City).
 Room (RoomID, HotelID)
 Customer (CustomerID, CustomerName)
 RoomAvailability (RoomID, HotelID, BeginDate, EndDate)
 Booking (BookID, HotelID, RoomID, CustomerID, BeginDate, EndDate)

Figure 5: Schema of the Room Booking database (from [Sali 01])

A.1.2. Relevant Class Diagrams

The logical structure of the user input described in Figure 2 and of the system output described in Figure 4 can be translated in a straightforward manner in corresponding UML Class diagrams. For instance, a class diagram for the output to the user related to the screen of Figure 4 is shown in Figure 6(a). Similarly, a Class Diagram including relationships can be developed to describe the database schema, as shown in Figure 6(b).

We note that the layout of the user interface, as shown in Figures 2 through 4, can not be described in UML. By the way, the situation is similar with the Lyee methodology. One assumes that the user interface layout is separately described. And suitable tools exist for generating automatically programs (in Visual Basic or Java, or as HTML pages) that realize the given interface layouts.

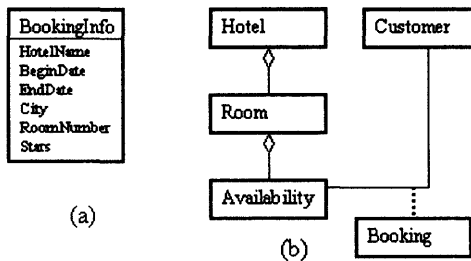


Figure 6: (a) The BookingInfo class corresponding to the output of the screen of Figure 4; (b) Class diagram representing the database schema of Figure 5 (without the class attributes)

A.1.3. Dynamic behavior requirements

The dynamic behavior of the Room Booking system can be described by the Activity Diagram shown in Figure 7. The activity *Prepare booking request* displays first the initial screen of Figure 2 and lets the user fill in the fields. The four subsequent actions correspond to the following four cases:

- The user pushes the *Quit* button.
- The customer identifier is not in the database.
- No suitable room is available.
- A room is available and may be reserved.

Case (4) leads to the activity *Confirm booking*, which displays the screen of Figure 4 and lets the user validate the reservation or go back to the initial screen.

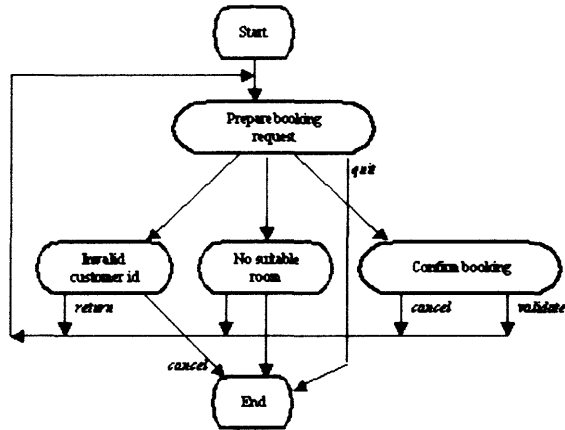


Figure 7: Activity diagram showing requirements on the dynamic behavior

We note that, in this example, each activity has the same sequence of interactions with the user: First a screen is presented to the user including values in certain information fields, and then the user may enter date into certain fields and pushes one of the buttons.

We also note that this activity diagram describes the behavior of the whole Room Booking system (including the database). In the case study of [Sali 01], the Lyee methodology is applied to specifying the Room Booking application program, which interacts with the database through SQL queries. The application, therefore, has one additional interface, namely the one for querying the database. If one wants to define the requirements for this application program, one has first to do some analysis in order to determine the type of SQL queries that are appropriate for this application. An activity diagram describing the dynamic requirements of the Room Booking application could then be obtained from the activity diagram of Figure 7 by refining the activities within the diagram and including the appropriate database queries and answers.

A.2. Development of the Room Booking example with Lyee methodology

In this subsection, we try to summaries the main steps of the case study described in [Sali 01]. The specification of the Room Booking application proceeds in two steps: first the aspects of data structures and interfaces and then the aspect of dynamic behavior (control flow). The first aspect is essentially covered by the definition of the so-called Logical Units. A Logical Unit corresponds to a screen or a database query and always includes input and output attributes. The Logical Unit and their attributes (called Words) for this application are shown in the table of Figure 8. Note that the last table column indicates the basic datatype of the Words (9 = numeric, X = alphanumeric, K = button); the input /output nature of the Words is not indicated in this table.

	ID	Name	Domain
<i>For screen1</i>	CustomerID	CustomerID	9
	BeginDate	BeginDate	X
	EndDate	EndDate	X
	Stars	Stars	9
	City	City	X
	CmdOK	CmdOK	K
<i>For screen2</i>	CmdQuit	CmdQuit	K
	HotelName	HotelName	X
	Begin	Begin	X
	End	End	X
	City	City	X
	RoomNum	RoomNum	X
<i>For screen3</i>	Stars	Stars	9
	CmdValid	CmdValid	K
	CmdCancel	CmdCancel	K
	Message1	Message1	X
	CmdReturn	CmdReturn	K
	CmdCancel	CmdCancel	K
<i>For screen4</i>	Message2	Message2	X
	CmdReturn	CmdReturn	K
	CmdCancel	CmdCancel	K
<i>For CustomerDB</i>	CustomerID	CustomerID	9
	CustomerName	CustomerName	X
<i>For Room Availability</i>	RoomID	RoomID	X
	HotelID	HotelID	X
	BeginDate	BeginDate	X
	EndDate	EndDate	X
<i>For BookingDB</i>	BookID	BookID	9
	RoomID	RoomID	X
	HotelID	HotelID	X
	CustomerID	CustomerID	9
	BeginDate	BeginDate	X
	EndDate	EndDate	X
<i>For AvailableRoom</i>	HotelName	HotelName	X
	HotelID	HotelID	X
	RoomID	RoomID	X

Figure 8 (from [Sali 01]): List of defined Domain Words

The dynamic aspects of the requirements are defined in the form of one of several so-called Process Route Diagrams (PRD), which are graphical representations of the control flow aspects of the application program. The PRD for the Room Booking application is shown in Figure 9.

One basic characteristic feature of the Lyee methodology is the standardized control structure which includes an internal loop within each of the W04, W02 and W03 Pallets and the loop repeating the execution of the Pallets within a given Scenario Function in the order W04, W02 and then W03 until no change occurs to the values of the variables associated with these Pallets. The PRD of Figure 9 includes 6 such Scenario Functions, named SF01 through SF06. From an abstract point of view, the PRD of Figure 9 defines the same control flow structure as the activity diagram of Figure 7; the Scenario Functions SF01 and SF02 correspond to the activity *Prepare booking request*, and SF03 and SF06 correspond to the *Confirm booking* activity. The other Scenario Functions SF04 and SF05 correspond to the other two activities shown in Figure 7. It is to be noted that the PRD, in addition, shows information about what items are input or output and the interactions with the database (since the application has an explicit interface with the database).

Automated Word Generation for the Lyee Methodology¹

Benedict AMON
Love EKENBERG
Paul JOHANNESSON
Marcelo MUNGUANAZE
Upendo NJABILI
Rika Manka TESHIA

Dept. of Computer and Systems Sciences
Stockholm University and KTH
Forum 100
SE-164 40 Kista
SWEDEN

Dept. of Information Technology and Media
Mid Sweden University
SE-851 70 Sundsvall
SWEDEN

Abstract. A conceptual schema can be viewed as a language to describe the phenomena in a system to be modelled, i.e., a set of derivation rules and integrity constraints as well as a set of event-rules describing the behaviour of an object system. In this paper, we are investigating the relationship between the Lyee software requirements concepts with various constructs in conceptual modelling. Within our work we choose the Unified Modelling Language (UML) as a modelling notation for the purpose of explaining conceptual models. The result obtained models a fully expressive set of UML and First Order Logic constructs mapped into Lyee concepts at the meta level. In particular, we look at the mapping of both the static and dynamic parts of UML concepts and structures to First Order Logic and LyeeALL input requirements.

1. Introduction

Traditionally, software systems were classified as either administrative, data intensive information systems, e.g. airline booking systems, or technical systems with complex computations with little data, e.g. process control systems. However, software systems are increasing rapidly, both in number and in complexity. Novel application areas, having been a vision yesterday, are today the reality, e.g. electronic commerce, education via world wide web, and global distribution of objects to mention a few. Embedded systems are another kind of

¹ This work was supported by the Lyee International Project.

software applications, rapidly increasing in number, e.g. components in a car or in a hi-fi equipment. This implies, among other things, that the demands of software engineering are ever increasing.

Quality software is supposed to satisfy the stated and implied intention of a customer. When discussing the quality of the software, three items are of vital concern: *the customer intention*, *the requirements* and *the code*. The requirements must capture the stated and implied user intention, and the software must meet the requirements, i.e. to function according to the requirements. It should also obey any non-functional requirements concerning e.g. usability, availability, cost for development and operation. The activity of assuring that the requirements capture the customer intention is usually referred to as *validation*. The activity of assuring that the software meets its requirements is usually referred to as *verification*. In other words, the validation is the activity linking the requirements to the intention, while verification links the requirements to the implementation.

Formal methods have mainly been advocated for technical types of systems, and safety critical applications. The tradition has been reflected in industrial development, research, and education. However, the rapid technological development, and the increasing number and complexity of software systems are gradually padding the borders – the systems become more complex, even the data intensive systems. Furthermore, technical systems need more data, but above all, customers require that also systems, not labelled as safety critical in the original meaning of the term, should function well and without interruption. The costs of interruption are considered too high even if not measured in human lives, but in money.

However, formal methods in software engineering are merely a partial solution to the quality problem. Formal methods at the current level of development are said to be at the stage that programming languages were in the early 1960's: cryptic, requiring much manual tinkering, with little tool support. Programmers are an elite group who used to talk in cryptic mathematical notations.

The Lyee methodology is a way of handling verification problems. Given that a user is able to state a set of requirements, the program can be generated in an interactive way until the requirements are satisfied. This makes the verification process a component embedded in the program development process. One crucial issue, however, is the validation process, i.e., to what extent do the stated requirements reflect the actual user intentions and the goals of the customer organisation. At present the Lyee methodology provides no conclusive support for determining this.

Several models and representation forms need to co-exist in order to enable users to get a good understanding of a system. Conceptual Models may be expressed in different notations, such as ER, UML, ORM, and other formalisms. These notations are typically graphic based and make use of intuitive and visual modelling constructs as the main components of the notations, which facilitates their adoption among large user groups.

However, the constructs of many conceptual modelling approaches are typically informally defined, which leaves room for ambiguities, loose interpretations and misunderstandings. An important task is, therefore, to formalise the basic notions and constructs of conceptual modelling. An attractive way to do this is to translate them into a some suitable logic representation. A logic representation allows for different kinds of analysers that can be used to verify and validate the models.

Conceptual modelling and logic representations can be used as means for capturing, verifying, and validating user requirements. However, they are not sufficient for generating executable systems. There is a need for a framework that is able to transform user requirements to executable software, which is exactly the strength of Lyee. Thus, we get a relationship

between conceptual models, first order logic, and Lyee constructs.

Against this background, the objective of the work of the Sweden Unit is to design techniques and methods for bridging the gap between customer intention and stated requirements within the framework of Lyee. For this purpose, the Unit are working with the development of the following components:

- A formal requirement language
- Verification techniques and supporting technology
- Validation techniques and supporting technology
- Translation algorithms between different formalisms and Lyee constructs

The overall architecture of the work is depicted in Fig. 1.

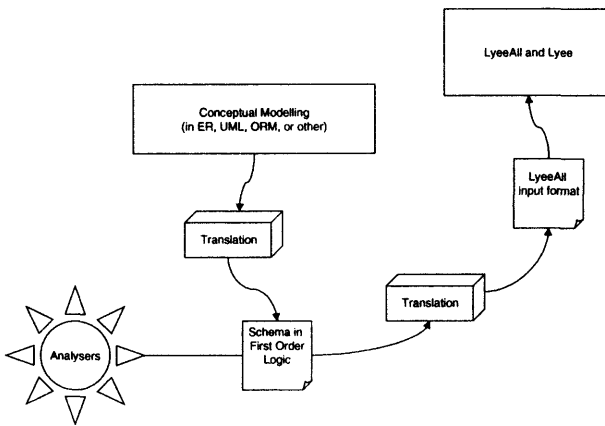


Fig. 1 Overall Architecture

Consequently, a main goal of the Sweden Unit is a tool for bridging the gap between the user intention and the actual program generation. To achieve this goal, there is a need for an interface for generating the input of words to the LyeeAll environment. Instead of reinventing such an interface, we have decided to use a subset of the Unified Modeling Language, UML [1] constructs for this purpose. This is not because we believe that UML is the only choice for representing conceptual models, but rather because there are adequate tools, such as Rational Rose [2] or Microsoft Visio [3] for stating UML models. Furthermore, UML has gained increased popularity in recent years and is now regularly used not only for systems analysis and design, for which it was originally conceived, but also for other phases in the systems life cycle such as requirements engineering and business analysis. The success of UML can to a large extent be attributed to two factors. First, UML has received extensive industry support from IT suppliers as well as users and has been effectively standardised. Secondly, UML makes use of intuitive and visual modelling constructs as the main components of the language, which facilitates its adoption among large user groups.

The paper is organised as follows. In section 2, we introduce the mapping of Conceptual

Models concepts into Lyee. In section 3, we show how first order logic notions can be related to Lyee constructs. In section 4, we show how to map between first order logic notions and Conceptual Models concepts. In the final section, we conclude the paper.

2. Lyee and Conceptual Models

In this section, we propose a mapping between Lyee and conceptual models. The mapping shows how conceptual modelling constructs can be realised in Lyee. This realisation results in an application that is capable of managing the structures and behaviours of the conceptual models.

2.1 An overview of Requirements for LyeeAll Tool

Lyee is a methodology to develop software by simply defining the software requirements. These requirements are defined by words, the formulae that should regulate the words, conditions to form each of these formulae, and the layout of screens and printouts. Once the requirements are defined, Lyee leaves all subsequent processing to algorithms based on the definitions [4], [5], [6].

Since an understanding of LyeeAll input requirements is needed to express the relationship between UML notions and LyeeAll input requirements, we will briefly explain the LyeeAll data elements before relating the UML notions to these.

Using the LyeeAll tool, the data can either be entered directly in the tool or imported from text files on the required LyeeAll format.

2.1.1 Defined and Items Data

When creating a system, the first thing a LyeeAll user needs to do is to enter the *Defined* and the corresponding *Items*. A *Defined* corresponds to a window or a database access definition. *Defineds* are composed of *Items* (where one *Item* belongs to at most one *Defined*), e.g. a database field or a button on a screen.

2.1.2 Process Route Diagram (PRD)

After identifying and entering *Defineds* and *Items* data, the next step is the creation of the *Process Route Diagram*.

A *Process Route Diagram* specifies the navigation between the various components of the application. This can be considered as a graph with a unique start node, intermediate nodes and one or several ending nodes. The nodes are called *Scenario functions*.

Each scenario function has three *Pallets* described in the following sections. Fig. 2 below shows the pallets and the order of their execution.

- W04 – corresponds to the display of information to screens or databases
- W02 – corresponds to the recovery of input from the screens or databases
- W03 – corresponds to the conditions for displaying information.

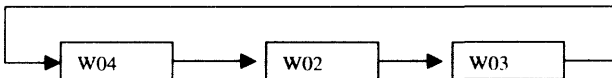


Fig. 2 A scenario function with its three Pallets in their order of execution

2.1.3 Pallets and Logical Units

Pallets are composed of *Words* grouped into *Logical Units*. This means that we will be specifying/implementing pallets when dealing with Logical Units. In other words Pallets are implemented using Logical Units.

2.1.4 Specification of Words in Logical Units

The relationship between Words and Logical Units is the same as the one between Items and Defineds. While Logical Units implement the behaviours of Defineds, Words implement the behaviour of Items.

2.1.5 Specification of Vectors

Vectors can also be regarded as Words. There are two kinds of vectors

1. Signification Vectors are composed of Domain Words already discussed above.
2. Action Vectors are of four kinds as follows:
 1. Input Vectors – implement input actions e.g. reading the screen
 2. Output Vectors – implement output actions e.g. displaying information to a screen
 3. Structural Vectors – implement the initialization of screen elements such as fields or buttons
 4. Routing Vectors – Implement the navigation between the application component

To summarize, Fig. 3 below shows the data elements for the LyeeAll input requirements and the steps to be followed in entering data into the LyeeAll tool.

In Fig. 3, the numbering of the data elements suggests the order to be followed in entering data. For instance, you need to enter a Defined first before entering its corresponding Items; you also need to have a Process Route Diagram and then enter its Scenario Functions, followed by Logical Units and then Domain Words.

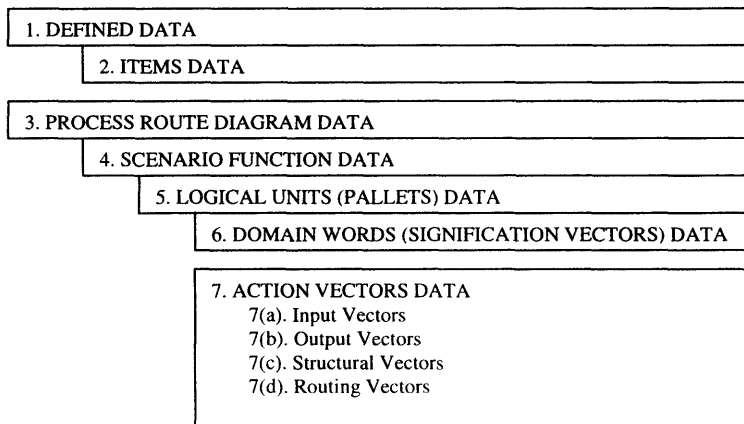


Fig. 3 Data Elements in LyeeAll Requirements

2.2 Translating static UML notions into LyeeAll Requirements

In this section, we suggest a translation of the static properties of class diagrams into LyeeAll requirements.

Fig. 4 shows the suggested relationship between static UML notions and LyeeAll requirements. As can be seen, there are two layers in the figure; the UML notation layer (UML Layer) and the LyeeAll requirement layer (Lyee Layer).

The elements of the two layers are related to each other as well as to the elements in the same layer. For instance a *Class* in the UML Layer corresponds to a *Defined* in the Lyee Layer. Furthermore, in the Lyee Layer a *Defined* is composed of one or more *Items*; a *Class* can have one or more *Attributes*.

The UML layer includes also the concepts of state, transition, and guard (condition) and a state is linked to a Logical Unit in the Lyee layer. An ISA relationship concept is also included specifying generalization relationships between classes.

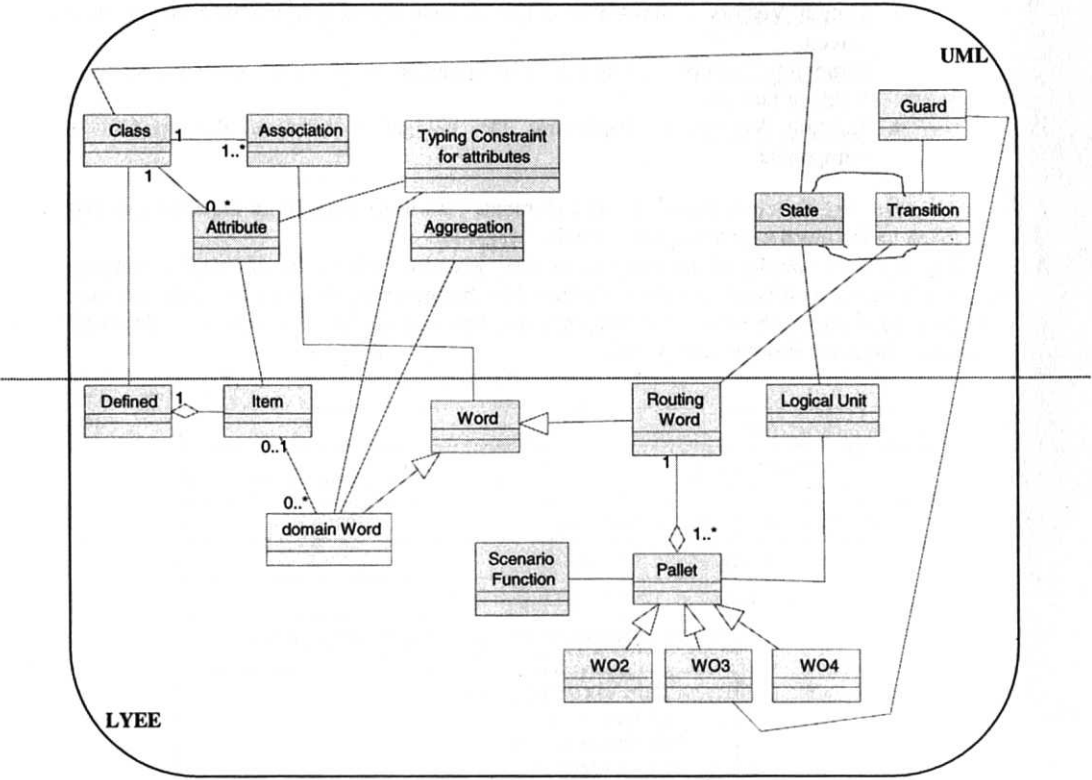


Fig. 4 Relationship between UML notions and LyeeAll Requirement

The following paragraphs explain the reasons for the relationships in Fig. 4.

1. Class definition

We have related a class in UML to a Defined in LyeeAll Requirements, as they both can function as containers for other constructs. Furthermore, corresponding data can also be related, e.g. an Attribute in a Class can be related to an Item in a corresponding Defined. Also in creating a user interface application, a likely UML object to be related to a screen (window) is a Class and if we look at LyeeAll, a Defined corresponds to a screen (window) or a database access.

However, not every class will give rise to a new Defined. This point is explained below.

2. ISA relationship

In case there is an ISA relationship in a UML diagram, a different approach to dealing with classes is required. In this approach, only one Defined is introduced for both classes.

As an example, consider the following scenario: There are two classes, say class1 and class2, in a UML class diagram and class1 ISA class2. In this case, class1 takes the attributes, operations, relationships and semantics of class2 and therefore only class1 is implemented as a Defined in LyeeAll.

3. Attributes

Attributes in classes of UML diagrams are linked to Items in Defineds. Since a class in UML is related to a Defined in LyeeAll (see classes above), then an attribute in a class can be related to an Item of a corresponding Defined. In the case where there is an ISA relationship between two classes in a UML diagram, say class1 ISA class2, then the attributes of class2 are considered as the attributes of class1 (see ISA relationship above)

4. Associations

Just as classes can relate with one another in an association, Defineds can be related to each other using words. For instance a routing word can be used to link one Defined to another Defined.

Below are a number of definitions establishing the relationship between static UML notions and LyeeAll requirements.

Definition 1

Given a class diagram C in a UML specification, R_C is the smallest set of LyeeAll constructs defined by the following clauses.

1. Class Definition

If r is a name of class definition in C , and for no q , q ISA r occurs in C , then r is a Defined in R_C .

2. Associations and Aggregations

If r and s are names of class definitions in C , and t is a name of an association from r to s in C , then $r.t.s$ is a word in R_C .

3. Attributes

If r is a name of a class definition in C and t is a name of an attribute of type integer (number) (string) of r in C , and for no q , q ISA r occurs in C then $r.t$ is an Item/domain word of data type 9 (9) (X) in R_C .

4. ISA constraints

If r and s are names of class definitions in C , and the statement r ISA s belongs to C , and t is an attribute of s of type integer (number)(string), then $r.t$ is an Item/domain word of data type 9 (9) (X) in R_C .

Example

Consider the UML class diagram in Fig. 5.

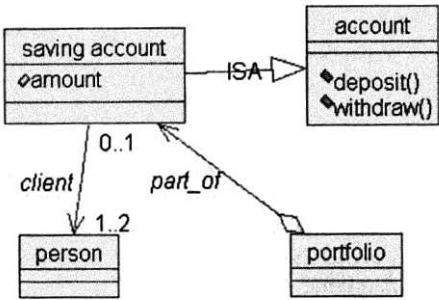


Fig. 5 A UML class diagram

Fig. 5 shows a UML class diagram. In the figure, *saving_account*, *account*, *person* and *portfolio* are classes in UML, representing concepts. An association represents the relation *client*, from *saving_account* to *person*, meaning that a client is a person with a saving account. The attribute *amount* represents the current balance. A person may own a saving account, and a person can have at most one saving account. Furthermore, at maximum two clients can share a saving account. The class *saving_account* is a subclass to *account*. Furthermore, *saving_account* is a component of the portfolio of the bank. This is represented by the aggregation form *saving_account* to *portfolio*. The methods *deposit* and *withdraw* represent possible transactions for an account and will be treated in dynamic part of this section.

Using definition 1, a class diagram above (Fig. 5) can be translated into the following LyeeAll Requirements (see Table 1 below).

Table 1 UML notations translate into LyeeAll

UML components	Translation of static properties
Class Person	Defined <i>person</i>
Class Portfolio	Defined <i>portfolio</i>
Class Account, Class Saving_account, and ISA relation	Defined <i>saving_account</i>
Association <i>client</i>	Word <i>saving_account.client.person</i>
Attribute <i>amount</i>	Item <i>saving_account.amount</i>
Aggregation <i>part_of</i>	Word <i>saving_account.part_of.portfolio</i>

2.3 Translating dynamic UML notations into LyeeAll Requirements

In UML, dynamics can be modelled in three different ways: by methods in classes, by statechart diagrams, and by interaction diagrams. In this section, we suggest the translation of statechart diagrams.

2.3.1 Translating Statechart diagrams into LyeeAll Requirements

Fig. 6 shows events and states of a saving account class. Transitions are shown as arrows, labelled with their events. States are shown in rounded rectangles.

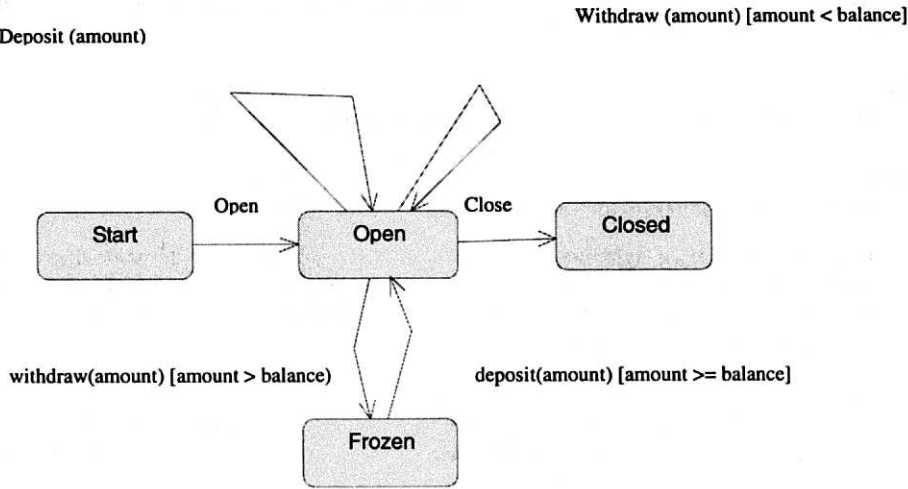


Fig. 6 A statechart diagram

The figure shows that the *saving_account* class can have four states, i.e. start, open, frozen, and closed. The events are open, deposit, withdraw, and deposit. It can also be noted that the conditions for the events are put inside the square brackets beside the corresponding events.

The states, transitions and their conditions can all be translated into LyeeAll requirements as explained below.

A state in UML is linked to a Logical Unit (pallet) in LyeeAll Requirements. While a Logical Unit in LyeeAll requirements describes the behaviour of a Defined; a class in UML changes states in response to events. For instance a *saving_account* changes from *Open* state to *Frozen* state in response to withdrawing below the minimum balance (*withdraw* event).

It is possible to implement a single Logical Unit for one or more states, e.g. a Logical Unit implementing the W04 pallet for writing new account data to the database could also be used to remove account data from the database.

A transition is a relationship between two states that indicates that when an event occurs, the object moves from the prior state to the subsequent state. An event is a significant or noteworthy occurrence [7].

A transition does not result in a state change by itself i.e. it represents the space between two states, and the event is responsible in activating the change. Due to this, in a LyeeAll realization of the UML transition/event, the event (and its guards) that causes the state change must be realized rather than the transition.

Events in UML can be considered as methods/formulas that, when occurring, result in a state change of a class. In LyeeAll, the data element that relates best to methods or formula is the Word. Because of this an event in UML is linked to a Logical Unit in LyeeAll Requirements.

Below are the natural language statements demonstrating the relationship between dynamic UML notions and LyeeAll requirements.

Definition 2

Given a state chart diagram S in a UML specification, R_S is the smallest set of LyeeAll constructs defined by the following clauses.

1. State

If r is a name of a state in S , then r is a Logical Unit in R_S .

2. Event

If r is a name of an event in S , then r is a Logical Unit in R_S .

Formally, a state is merely a signature. In the case of events, we are taking the same approach by just taking into account the signature of the event. However, the actual content of transitions is straightforwardly modelled in the respective pallets, depending of the character of the transition. The details will be omitted here.

Example

Considering a statechart diagram above (see Fig. 6) as an example and using definition 2 above, the statechart diagram is translated into the following LyeeAll Requirements, see table 2 below.

Table 2 Statechart Translation Example

UML components	Translation into LyeeAll
State <i>start</i>	Logical unit <i>start</i>
State <i>open</i>	Logical unit <i>open</i>
State <i>frozen</i>	Logical unit <i>frozen</i>
State <i>closed</i>	Logical unit <i>close</i>
Event <i>open</i>	Word <i>open</i>
Event <i>close</i>	Word <i>close</i>
Event <i>deposit</i>	Word <i>deposit</i>
Event <i>withdraw</i>	Word <i>withdraw</i>

In table 2, the Words *deposit* and *withdraw* will also contain the condition to check that the amount (current balance) is greater/equal to the balance (minimum balance) or otherwise trigger other events/words.

3. Lyee and First Order Logic

In this section, we propose a mapping between Lyee and First Order Logic (FOL). The purpose of this mapping is to make explicit how constructs of Lyee are related to FOL constructs. We start by introducing two definitions that show how typical conceptual modelling constructs, static as well as dynamic, can be represented in FOL extended with an event concept.

Definition 3

A *schema* *S* is a structure $\langle R, ER \rangle$ consisting of a *static part* *R* and a *dynamic part* *ER*. *R* is a finite set of closed first-order formulae in a language *L*. *ER* is a set of *event rules*. Event rules describe possible transitions between different states of a schema and will be described below. *L(R)* is the *restriction of L to R*, i.e. *L(R)* is the set $\{p \mid p \in L, \text{ but } p \text{ does not contain any predicate symbol, that is not in a formula in } R\}$. The elements in *R* are called *static rules* in *L(R)*. A *static assertion* in *S*, *F(x)*, is a closed first order formula in *L(S)*.

Definition 4

An *event rule* in *L* is a structure $\langle P(\mathbf{z}), C(\mathbf{z}) \rangle$. *P(z)* and *C(z)* are first-order formulae in *L*, and *z* is a vector of variables in the alphabet of *L*.² In terms of conceptual modelling, *P(z)* denotes the precondition of the event rule, and *C(z)* the post condition.

Fig. 7, presents a meta-model that depicts the concepts and relations in the Lyee layer together with the main constructs in a FOL layer. The model also shows, on a superficial level, the relations between the two layers.

²The notation *A(x)* means that *x* is free in *A(x)*.

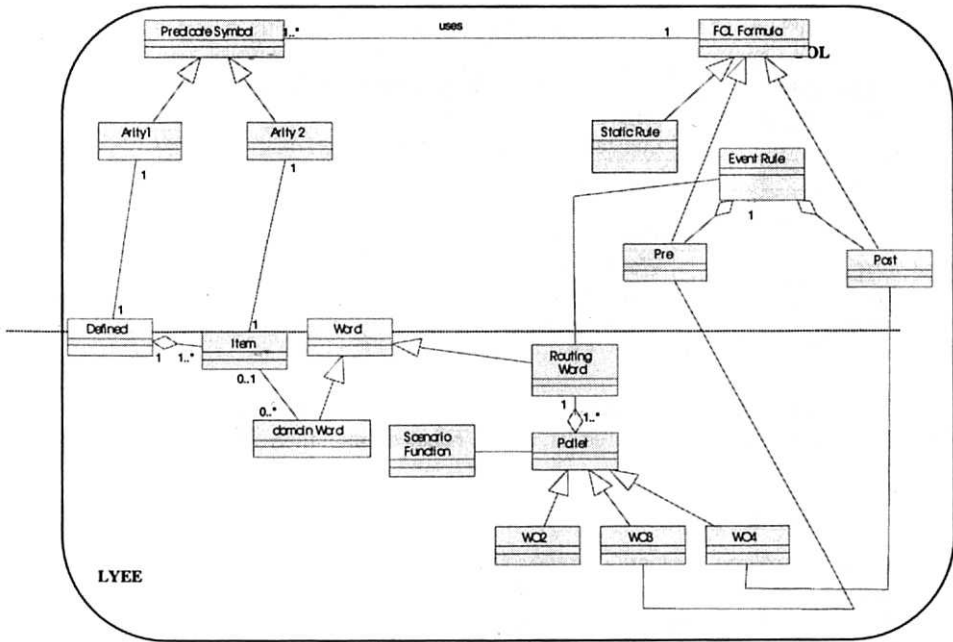


Fig. 7 Meta-Model showing the relationship between FOL and LYEE

The Lyee layer complies with the LyeeAll input requirements. Concepts such as Defined, Scenario Function, Domain words, Routing words, and Pallets occur at this layer. The concept of a routing word is used to distribute the control over the various scenario functions of a process route diagram, [6]. In particular, routing words are in charge of handing over the control from one pallet to another. The FOL layer contains constructs such as predicate symbols and FOL formulas.

The two layers are related to one another through a number of links. It can be seen that a Defined at the Lyee layer corresponds to a predicate of arity one at the FOL layer. The reason for this is that a Defined can exist independently of any other constructs, and it can therefore be represented by a simple one-place predicate. An Item at the Lyee layer corresponds to a predicate of arity two, since an Item is related to two constructs: a Defined and a type. A routing word corresponds to an event rule, as routing words manage the control flow in a process route diagram. A scenario function is a generic structure that allows the control of Lyee programs in a systematic way. The structure is generic, as it is capable of accommodating any kind of program dynamics [6]. W03 is the pallet in Lyee concerned with evaluating conditions that must hold for the calculation to be performed. This has been related to a pre-condition at the FOL layer, since the latter consists of a condition that must be satisfied before any action is performed on some variable. W04 at the Lyee layer is concerned with calculating and writing output data on a medium (screen or database) after it has been passed to pallet W03, and the condition has been satisfied. We, thus, link this with the post condition at the FOL layer. Below, we write down the mapping introduced more formally.

Definition 5

Given a set L of LyeeALL constructs, R_L is the smallest set of first order formulae defined by the following clauses:

Alphabet

1. If r is a predicate symbol of arity one in R_L , then r is a Defined in L .
2. If r is a predicate symbol of arity two in R_L , then r is an Item in L .

Constraints

3. If r is a precondition in R_L , then r is a constraint in pallet W03 in L .
4. If r is a post condition in R_L , then r is constraint in pallet W04 in L .

4 Translating Conceptual Models into First Order Logic

Following [8], static and dynamic concepts modelled in UML can straightforwardly be expressed in terms of FOL formulae. In this section, we suggest such a translation for a relevant subset of the UML constructs. Fig. 8 illustrates the mapping of UML constructs to FOL formulae.

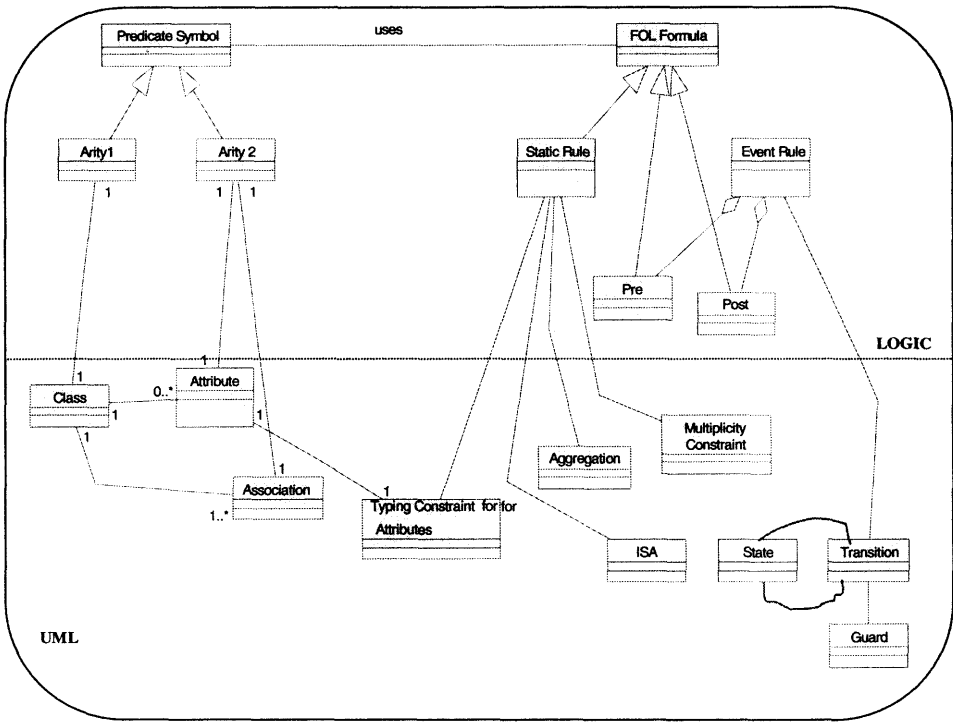


Fig. 8 Meta Model showing the Relationship between FOL and UML

4.1 Translating Class Diagrams into Logic

Definition 6

Given a set C of class diagrams in a UML specification, where the strings *agg* or *lex* do not occur. R_C is the least set of first order formulae defined by the following clauses.

1. Alphabet

- If r is a name of a class definition in C , then r is a predicate symbol of arity one in $L(R_C)$.
- If t is a name of an association in C , then t is a predicate symbol of arity two in $L(R_C)$.
- If t is a name of an attribute in C , then t is a predicate symbol of arity two in $L(R_C)$.
- agg is a predicate symbol of arity two in $L(R_C)$.
- lex is a predicate symbol of arity one in $L(R_C)$.

2. Typing constraints for associations

If r and s are names of class definitions in C , and t is a name of an association from r to s in C , then $\forall x \forall y (t(x,y) \rightarrow (r(x) \wedge s(y)))$ is in R_C .

3. Typing constraints for attributes

If r is a name of a class definition in C and t is a name of an attribute of r in C , then $\forall x \forall y (t(x,y) \rightarrow (r(x) \wedge lex(y)))$ is in R_C .

4. Aggregation constraints

If r and s are names of class definitions in C , and t is a name of an aggregation from r to s in C , then $\forall x \forall y (t(x,y) \rightarrow (r(x) \wedge s(y) \wedge agg(x,y)))$ is in R_C .

5. ISA constraints

If r and s are names of class definitions in C , and the statement r ISA s belongs to C , then $\forall x (r(x) \rightarrow s(x))$ is in R_C .

6. Subclass constraints

Assume that p , r and s are names of class definitions in C , and that p ISA s and r ISA s belong to C . If p and r are disjoint in C , then $\forall x \neg (p(x) \wedge r(x))$ is in R_C . If p and r are exhaustive wrt s in C , then $\forall x (s(x) \rightarrow (p(x) \vee r(x)))$ is in R_C .

7. Cardinality constraints

If r and s are names of class definitions in C , and t is a name of an association from r to s in C , with cardinality $((\min_r .. \max_r), (\min_s .. \max_s))$, then the formulae below are in R_C .

7.1. Minimum number of associations for the domain

$$\begin{aligned} & \forall y \exists x_1 \dots \exists x_{\min_r} ((s(y)) \rightarrow (t(x_1, y) \wedge \dots \wedge t(x_{\min_r}, y))) \wedge \\ & \neg(x_1 = x_2) \wedge \dots \wedge \neg(x_1 = x_{\min_r}) \wedge \\ & \neg(x_2 = x_3) \wedge \dots \wedge \neg(x_2 = x_{\min_r}) \wedge \dots \wedge \\ & \neg(x_{\min_r-1} = x_{\min_r}) \end{aligned}$$

7.2. Maximum number of associations for the domain

$$\begin{aligned} & \forall y \forall x_1 \dots \forall x_{\max_r} \forall x_{\max_r+1} [((t(x_1, y) \wedge \dots \wedge t(x_{\max_r}, y) \wedge t(x_{\max_r+1}, y)) \\ & \rightarrow \\ & ((x_1 = x_2) \vee \dots \vee (x_1 = x_{\max_r}) \vee (x_1 = x_{\max_r+1}) \vee \\ & (x_2 = x_3) \vee \dots \vee (x_2 = x_{\max_r}) \vee (x_2 = x_{\max_r+1}) \vee \dots \vee \\ & (x_{\max_r} = x_{\max_r+1})))] \end{aligned}$$

7.3. Minimum number of associations for the range

$$\begin{aligned} & \forall y \exists x_1 \dots \exists x_{\min_s} ((r(y)) \rightarrow (t(y, x_1) \wedge \dots \wedge t(y, x_{\min_s}))) \wedge \\ & \neg(x_1 = x_2) \wedge \dots \wedge \neg(x_1 = x_{\min_s}) \wedge \\ & \neg(x_2 = x_3) \wedge \dots \wedge \neg(x_2 = x_{\min_s}) \wedge \dots \wedge \\ & \neg(x_{\min_s-1} = x_{\min_s}) \end{aligned}$$

7.4. Maximum number of associations for the range

$$\begin{aligned} & \forall y \forall x_1 \dots \forall x_{\max_s} \forall x_{\max_s+1} [((t(y, x_1) \wedge \dots \wedge t(y, x_{\max_s}) \wedge t(y, x_{\max_s+1})) \\ & \rightarrow \\ & ((x_1 = x_2) \vee \dots \vee (x_1 = x_{\max_s}) \vee (x_1 = x_{\max_s+1}) \vee \\ & (x_2 = x_3) \vee \dots \vee (x_2 = x_{\max_s}) \vee (x_2 = x_{\max_s+1}) \vee \dots \vee \\ & (x_{\max_s} = x_{\max_s+1})))] \end{aligned}$$

4.2 Methods

Definition 7

Given a set of methods M in a UML specification, where the string *inv* does not occur. ER_M is the least set of expressions defined by the following clauses.

1. Alphabet

- If $g(y) = \langle \text{pre}(x), \text{post}(x) \rangle$ is a method in M , then the corresponding predicate symbols with the same arities are in $L(ER_M)$.
- inv* is a predicate symbol of arity one in $L(ER_M)$.

2. Methods

Let k be a class and $g(y) = \langle \text{pre}(x), \text{post}(x) \rangle$ a method in k . If $g(y)$ is a method in M , Then $\langle \text{inv}(g(y)) \wedge \text{pre}(x), \text{post}(x) \wedge \neg \text{inv}(g(y)) \rangle_k$ is in ER_M , where $\text{pre}(x)$ and $\text{post}(x)$ are as above.

4.3 State Diagrams

Definition 8

A *state diagram* for a class k is a triple $\langle N, A, G \rangle$, where

- N is a set of nodes.
- A is a set of directed arcs, i.e. pairs over N .
- G is a set of triples $\langle a, g(y), v(x) \rangle_k$, where a is an arc, $g(y)$ is the signature of a method in k , and $v(x)$ is an open first order formula.

Definition 9

Let $\langle N, A, G \rangle$ be a state diagram for a class k . Let $\langle \langle t_i, t_j \rangle, g(y), v(x) \rangle \in G$. The *arc event* of $\langle t_i, t_j \rangle$ is $\langle \text{inv}(g(y)) \wedge \text{state}(x, t_i) \wedge v(x), \text{state}(x, t_j) \wedge \neg \text{inv}(g(y)) \rangle_k$.

Definition 10

Given a set S of state diagrams $\langle N, A, G \rangle_k$ in a UML specification, where the string *state* does not occur. A *schema* for S is constructed as follows.

1. Alphabet

- If $\langle \langle t_i, t_j \rangle, g(y), v(x) \rangle \in G$, then the corresponding predicate symbols and constants in $t_i, t_j, g(y)$ and $v(x)$ with the same arities are in $L(ER_S)$ and $L(R_S)$.
- state* is a predicate symbol of arity two in $L(ER_S)$ and $L(R_S)$.

2. Rules for an object in a state

$R_S = \{ \forall x(\text{state}(x, t_i) \rightarrow \neg \text{state}(x, t_j)) \mid t_i, t_j \in N \wedge i \neq j \} \cup \{ \forall x \exists t(\text{state}(x, t)) \} \cup \{ \forall x(\text{state}(x, t_i) \rightarrow k(x)) \mid t_i \in N \}$, where t_i is a state in a class k .

3. Arcs

If $\langle \langle t_i, t_j \rangle, g(y), v(x) \rangle$ is an arc in G , $\langle \text{inv}(g(y)) \wedge \text{state}(x, t_i) \wedge v(x), \text{state}(x, t_j) \wedge \neg \text{inv}(g(y)) \rangle_k$ is in ER_S .

4. Schema

The *schema* for S is the structure $\langle R_S, ER_S \rangle$.

4.4 Collaboration Diagrams

Definition 11

A *collaboration diagram* for a UML specification is a pair $\langle B, M \rangle$, where

- B is a set of objects.
- M is an ordered set of quadruples $m_i = \langle a, m(x), b, i \rangle$, where a and b are objects, $m(x)$ is a method between a and b and where i is a natural number.³

Intuitively, such a triple in M specifies a method from a to b . Furthermore, this is the i :th method in a collaboration diagram.

Definition 12

Given a collaboration diagram $D = \langle B, M \rangle$ in a UML specification, where the string *sent* and *method* does not occur. A *schema* for D is constructed as follows.

³ We assume that all methods have unique names. Furthermore, each method is assigned a new name for all times it is called. Since a collaboration diagram is finite, this does not imply a loss of generality.

1. Alphabet

- If $\langle a, m(x)_k, b, i \rangle \in M$, then the corresponding predicate symbols and constants with the same arities are in $L(R_D)$.
- sent is a predicate symbol of arity two in $L(R_D)$.

2. Static rules

- $\{\forall x(\text{inv}(x) \rightarrow \text{sent}(x))\} \in R_D$
- If $\langle y1, m_i(x), y2, i \rangle, \langle z1, m_j(x), z2, j \rangle \in M$, and $i < j$, then $\{\forall x(\text{sent}(m(x)_i) \rightarrow \text{sent}(m(x)_j))\} \in R_D$

3. Schema

The *schema for D* is the structure $\langle R_D, \emptyset \rangle$.

Definition 13

Let C be a set of class diagrams, M a set of methods, S a set of state diagrams, and D a collaboration diagram of a UML specification U . Furthermore, let R_C, R_S, R_D, ER_S and ER_M be as in the definitions above regarding these components. A schema for U is the structure $\langle R, ER \rangle$, where

- $R = R_C \cup R_S \cup R_D$, and
- $ER = ER_M \cup ER_S$.

Consequently, a structure $\langle R, ER \rangle$ constructed as in the definition above is a translation of a UML specification into a schema. From the construction, it follows that this schema has the same properties as the UML specification from which it was derived, considering the subset of UML constructs under consideration.

Example

Consider the UML class diagram in section 2, Fig. 5. Thus, *saving_account*, *account*, *person* and *portfolio* are classes in UML, representing concepts. An association represents the relation *client*, from *saving_account* to *person*, meaning that a client is a person with a saving account. The attribute *amount* represents the current balance. A saving account must be owned by a person, and a person can have at most one saving account. Furthermore, at maximum two clients can share a saving account. The class *saving_account* is a subclass to *account*. Furthermore, *saving_account* is a component of the portfolio of the bank. This is represented by the aggregation form *saving_account* to *portfolio*. The methods *deposit* and *withdraw* represent possible transactions for an account and will be treated in section 4 below.

Using the rules in definition 10, the class diagram is translated to the set of formulae in Table 3.

Table 3 Static Translation Example

UML components	Translation of static properties
ISA relation	$\forall x(\text{saving_account}(x) \rightarrow \text{account}(x)),$
Association <i>client</i>	$\forall x \forall y(\text{client}(x,y) \rightarrow (\text{saving_account}(x) \wedge \text{person}(y))),$
Attribute <i>amount</i>	$\forall x \forall y(\text{amount}(x,y) \rightarrow (\text{saving_account}(x) \wedge \text{lex}(y))),$
Aggregation <i>part_of</i>	$\forall x \forall y(\text{part_of}(x,y) \rightarrow (\text{saving_account}(x) \wedge \text{portfolio}(y) \wedge \text{agg}(x,y))),$
Cardinality 1	$\forall x \exists y(\text{saving_account}(x) \rightarrow \text{client}(x,y)),$
Cardinality ..2	$\forall x \forall y \forall z \forall w((\text{client}(x,y) \wedge \text{client}(x,z) \wedge \text{client}(x,w)) \rightarrow ((y=z) \vee (y=w) \vee (z=w))),$
Cardinality ..1	$\forall x \forall y \forall z((\text{client}(y,x) \wedge \text{client}(z,x)) \rightarrow (y=z))$

4.5 Translation Script

In this section we describe an implementation of the translation algorithm of UML concepts into first order logic formulae. The translation script is written in a basic script language using Rational Rose, a support modelling tool for UML. A user can design a model of a system using Rose. In order for the script to function the user should specify constructs such as class name, attributes, association, generalization and multiplicity. The script can be run from the Rose menu to automate the translation process. Existing concepts are translated. The mapping rules used in the script correspond to the algorithm described in the previous section.

5. Concluding Remarks

In this paper, we have investigated the relationship between LyeeAll software requirement concepts and other techniques for representing user requirements, taking into account static as well as dynamic features of the various constructs. In our work, we have chosen First Order Logic and the Unified Modelling Language (UML) as basic modelling notations for the representation of conceptual models.

More precisely, we have suggested a set of translation algorithms for meta-level mappings of the relationship between UML constructs, formulae in First Order Logic and LyeeALL input requirements. The obtained result is a mapping from a fully expressive subset of the UML as well as a transition logic into Lyee concepts at the meta level. We have investigated both how the static and dynamic parts of UML as well as how structures in First Order Logic can be mapped on LyeeALL input requirements. Furthermore, we have identified the set of concepts needed for LyeeALL to automatically generate a program fulfilling user requirements.

It should be emphasised that the choice of UML is not because we embrace it as the saviour of modern programming, but simply because it is a well-known formalism that has been implemented in several software packages. The choice of logic is even more natural, since it allows for elaborated verification and validation techniques.

References

- [1] Object Management Group (OMG). Unified language specification version 1.4, <http://www.uml.org>, 2002.
- [2] Rational. [Http://www.rational.com](http://www.rational.com), 2002.
- [3] Microsoft, <http://www.microsoft.com>, 2002.
- [4] F. NEGORO, "Intent Operationalisation For Source Code Generation," The proceeding of SCI 2001 and ISAS 2001, 2001.
- [5] Development Using Lyee A Case Study With LyeeAll. Technical Report, University Paris 1, C.R.I. October 2001.
- [6] Requirements Modeling in Lyee, Technical Report. University Paris 1, C.R.I. October 2001.
- [7] C. Larman, Applying UML and Patterns, Prentice Hall PTR 2002.
- [8] L. Ekenberg and P. Johannesson, "UML as a Transition Logic," to appear in Proceedings of 12th European-Japanese Conference on Information Modelling and Knowledge Bases, 2002.

Static Analysis on Lyee-Oriented Software

Mohamed Mejri, Béchir Ktari and Mourad Erhioui

Computer Science Department,

Laval University, Quebec, Canada.

E-mail: {mejri,ktari,erhioui}@ift.ulaval.ca

Abstract. Software development has been suffering, for many decades, from the lack of simple and powerful methodologies and tools. Despite the tremendous advances in this research field, the crisis has still not overcome and the proposed remedies are far from resolving the problems of software development and maintenance. Lately, a new and very promising methodology, called Lyee, has been proposed. It aims to automatically generate programs from simple user requirements.

The purpose of this paper is, in one hand, to give a short and technical introduction to the Lyee methodology. And, in the other hand, to show how some classical static analysis techniques (execution time and memory space optimization, typing, slicing, etc.) can considerably improve many aspects of this new methodology.

1 Introduction

Software development and maintenance has become an activity with a major importance in our economy. As computer comes into widespread use, this activity involves a big industry. Hundreds of billions of dollars are spent every year in order to develop and maintain software. Today, competition between actors of software development field is fiercer than ever. To keep in the race, these actors (companies) must keep productivity at its peak and cost at its bottom. They must also deliver products (software) having high qualities and deliver them in time. However, do the available tools and methodologies for software development suit properly the company needs?

Basically, the goal of the software development researches is to look for how to build better software easily and quickly. A large variety of methodologies and techniques have been elaborated and proposed, over the last 10 years, to improve one or many steps of the software development life cycle. Despite their considerable contributions, they have a big difficulty to find their way into widespread use. In fact, almost all of them fail to produce clearly understood and modifiable systems and their use still considered to be an activity accessible only for specialists with a very large array of competencies, skills, and knowledge. This, in turn, requests highly paid personal, high maintenance, and extensive checks to be performed on the software. For these reasons, companies are now more than welcome to any new methodology promising improvement in software development cycle and they are ready to pay the price.

Lyee [5, 6, 7] (governmental methodology for software providence) is one of the new and very promising methodology. Intended to deal efficiently with a wide range of software problems related to different field, Lyee allows the development of software by simply defining their requirements. More precisely, the user has only to give words, their calculation

formulae, their calculation conditions (preconditions) and layout of screens and printouts, and then leaves in the hands of a computer all subsequent troublesome programming process (control logic aspects). Despite its infancy, the results of its use have shown its tremendous potential. In fact, compared to conventional methodologies, development time, maintenance time and documentation volume can be considerably reduced (70 to 80%) [6]. Up to now, a primitive supporting tool called LyeeAll is available to developers allowing the automatic generation of code from requirements.

Nevertheless, as any new methodology, researches have to be led on Lyee to prove its efficiency, to find out and eliminate its drawbacks and improve its good qualities. Furthermore, the LyeeAll tool has to be more developed to make it more user-friendly.

Through this paper, we show how classical static analysis techniques can considerably contribute to analyze Lyee requirements (a set of words within their definitions, their calculation conditions and their attributes) in order to help their users understanding them, discover their inconsistency, incomplete and erroneous parts, and to help generating codes with better qualities (consume less memory and execution time). Basically, the used static analysis techniques are:

The remainder of this paper is organized as follows. In Section 2, we give a short and technical introduction to the Lyee methodology. Section 3 shows how static analysis techniques can contribute in the enhancement of this methodology. Section 4 shows how typing and other static analysis techniques can improve some aspect of the Lyee methodology. Section 5 introduces the Lyee Requirement Analyzer, a prototype that we have developed to implement some static analysis techniques. Finally, in Section 6, some concluding remarks on this work and future research are ultimately sketched as a conclusion.

2 Lyee Methodology

Most people who have been seriously engaged in the study and development of software systems agree that one of the most problematic tasks in this process is to well understand requirements and correctly transforming them into code. To solve this problem, the Lyee methodology propose a simple way to generate programs from requirements.

Although the philosophic principles behind the Lyee methodology are very interesting, in this section we focus only on some practical ideas useful to understand how to write software using this methodology and how look the codes that are automatically generated from requirements.

2.1 Lyee requirements

Within the Lyee methodology requirements are given in a declarative way as a set of statements containing words together with their definitions, their calculation conditions and their attributes (input/output, types, and other attributes omitted within this paper for the sake of simplicity), as shown in Table 1.

Let s_w be the statement defining the word w , then the requirements given in the Table 1, correspond intuitively, in a traditional programming language, to the code given in Table 2.

Within the Lyee methodology, the user has not to specify the order (control logic) in which these definitions will be executed. As shown in Table 1, despite the fact that the definition of the word a uses the word b , the statement s_b is given after the statement s_a . The control

Table 1: Lyee Requirements.

Word	Definition	Condition	Input/Output	...
		:		...
a	b+c	b*e>2	Output	...
c			Input	...
b	2*c+5	c>0	Output	...
e			Input	...
		:		...

Table 2: Statement Codes.

Statement	Code
s_a	if b*e>2 then a:=b+c; output(a); endif
s_c	input(c);
s_b	if c>2 then b:=2*c+5; output(b); endif
s_e	input(e);

logic part of the software will be, within the Lyee methodology, automatically generated, then reducing consequently programming errors and time.

2.2 Code Generation

From requirements in Table 1, we can automatically generate a program that computes the value of *a* and *b* and output them. This program, will simply repeat the execution of these instructions until a fixed point is reached, i.e. any other iteration will not change the value of any word as shown in Fig. 1.

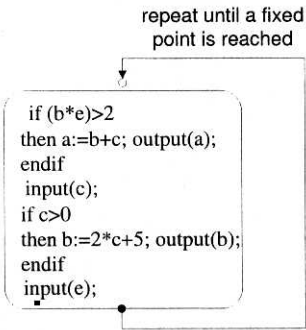


Figure 1: Requirement Execution.

Moreover, it is obvious that changing the order of code associated to the statement given in Table 2, the semantic of the program will never change, i.e. it will always associate the correct values to the words.

Let’s give more precision about the structure and the content of the program that will be automatically generated by Lyee from requirements. Within the Lyee methodology, the execution of a set of statements, such the ones given in Table 1, is accomplished in a particular manner. In fact, Lyee distributes the code associated to statements over three spaces, called *Pallets* (W02, W03 and W04) in the Lyee terminology, as shown in Fig. 2.

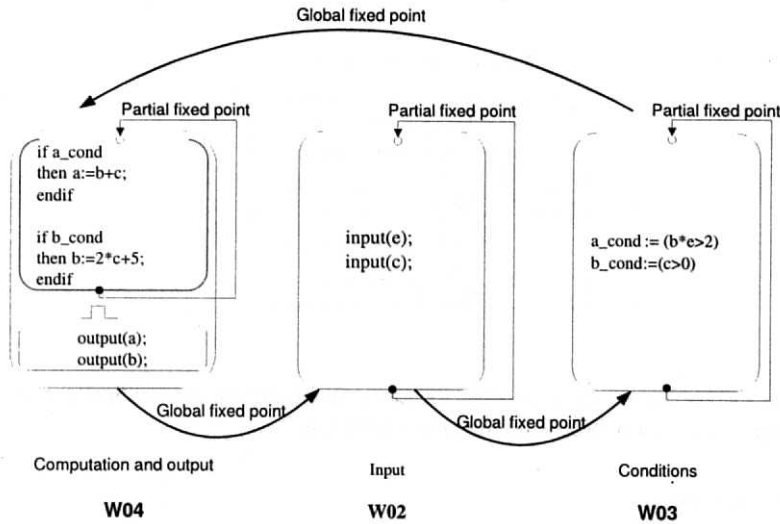


Figure 2: Lyee Pallets.

The pallet W02 deals with the input words, the pallet W03 computes the calculation conditions of the words and the results are saved in some boolean variables. For instance, the condition 'b*e>2' used within the definition of the word 'a' is calculated in W03 and the true/false result is saved in another variable 'a_cond'. Finally, the pallet W04 deals with the calculation of the words according to their definition given within the requirements. It also outputs the value of the computed words.

Starting from the pallet W04, a Lyee program tries to compute the values of all the defined words until a fixed point is reached. Once there is no evolution in W04 concerning the computation of the word values, the control is given to the pallet W02. In its turn, this second pallet tries repeatedly to input the missing words until a fixed point is reached (no others input are available) and then transfer the control to the pallet W03. Finally, and similarly to the pallet W04, the pallet W03 tries to compute the calculation conditions of the words according to the requirements until a fixed point is reached. As shown in Fig. 3, this whole process (W04 → W03 → W02) will repeat until a situation of overall stability is reached and it is called a Scenario Function. Besides, it is simple to see that the result of the execution of the program shown in Fig. 1 will be the same as the result of the one shown in Fig. 2.

In addition, Lyee has established a simple elementary program with a fixed structure

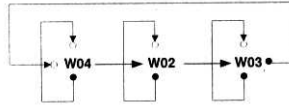


Figure 3: Scenario Function.

(called Predicate Vector in the Lyee terminology) that makes the structure of whole generated code uniform and independently from the requirement content. The global program will be simple calls of predicate vectors. The structure of a predicate vector is as shown in Fig. 4.

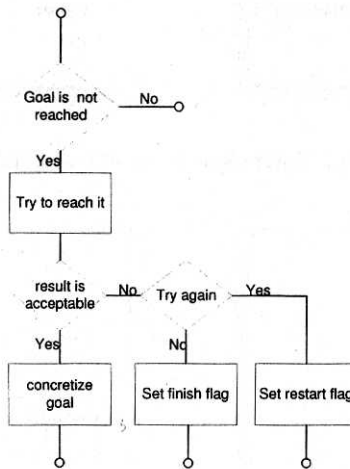


Figure 4: Predicate Vector.

The goal of a predicate vector change from one pallet to another. For instance, in the pallet *W04*, the first goal is to give a value to a word according to its calculation definition. For the example shown in Fig. 2, the predicate vectors associated to the calculation of the word 'a' and the word 'b' are as shown in Fig. 5.

Once there is no evolution in the calculation of the words, the Lyee generated code tries to output the words which will be the next goal. The predicate vector having the goal to output values is called output vector. In the pallet *W02*, we find two predicate vectors having as a goal to associate values to input words. For the sake of simplicity, predicate vector dealing with inputs, outputs and the initialization of the memory will be omitted within other specific details. Finally, in the pallet *W03*, the goal of predicate vectors is to compute preconditions specified within requirements as shown in Fig. 6.

Finally, the Lyee program associated to the requirements given in Table 1 is as shown in Table 3.

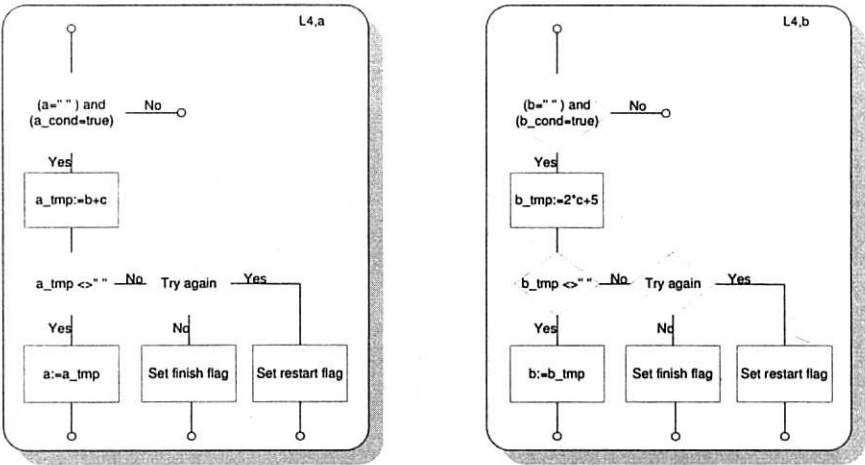


Figure 5: The Predicate Vectors of L4, a and L4, b.

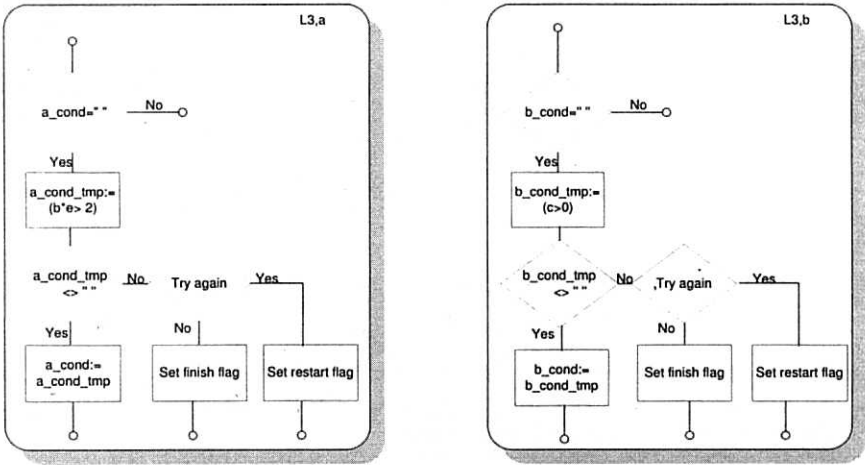


Figure 6: The Predicate Vectors of L3, a and L3, b.

2.3 Process Route Diagram

The Scenario Function presented in the previous section can be a complete program for a simple case of given requirements and specially when all the input and output words belong to the same screen and there is no use of any database. However, if we need to input and output words that belong to databases or to different screens interconnected together, then the situation will be a little complicated. For the sake of simplicity, we deal, in the sequel, only with the case when we have many screens. For instance, suppose that we have three

Table 3: Lyee Generated Program.

Pallet	Program	Comments
W04	Call S4 Do Call L4_a Call L4_b while a fixed point is not reached Call O4 Call R4	Initialize memory Calculate a Calculate b Output the result Go to W02
W02	Do Call L2_e Call L4_c while a fixed point is not reached Call I2 Call R2	 Input results Go to W03
W03	Do Call L3_a Call L3_b while a fixed point is not reached Call R3	Calculate a_cond Calculate b_cond Go to W04

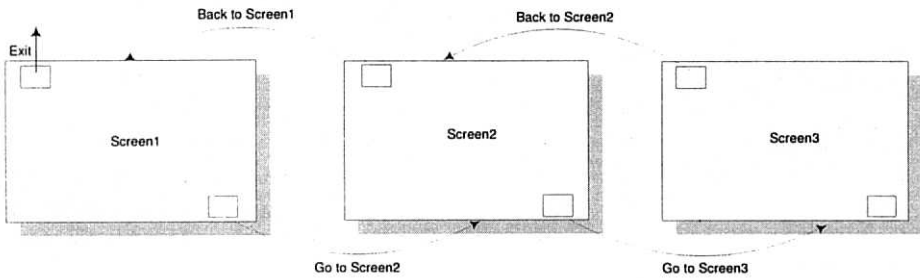


Figure 7: Screen Interactions.

interconnected screens, as shown in Fig. 7 allowing a user to navigate from one to another and in each one of them he can input, compute and output some words. Therefore, in the specification, the user has to give how these screens are interconnected.

Furthermore, it is not convenient to define only one scenario function in which we compute all the words defined in all the screens. In fact, some screens may not be visited for a given execution of the program and then the computation of the value of their words will be a lost of time. For that reason, Lyee associates to each screen its owner scenario function that will be executed only if this screen is visited. The scenario functions associated to screens are connected together showing when we move from one of them to another. In the Lyee terminology, many scenario functions connected together make up a Process Route Diagram as shown in Fig. 8.

To sum up, according to the Lyee methodology, generally a program contains many process route diagrams. Each of them is a set of interconnected scenario functions and each scenario function contains three interconnected pallets W02, W03 and W04.

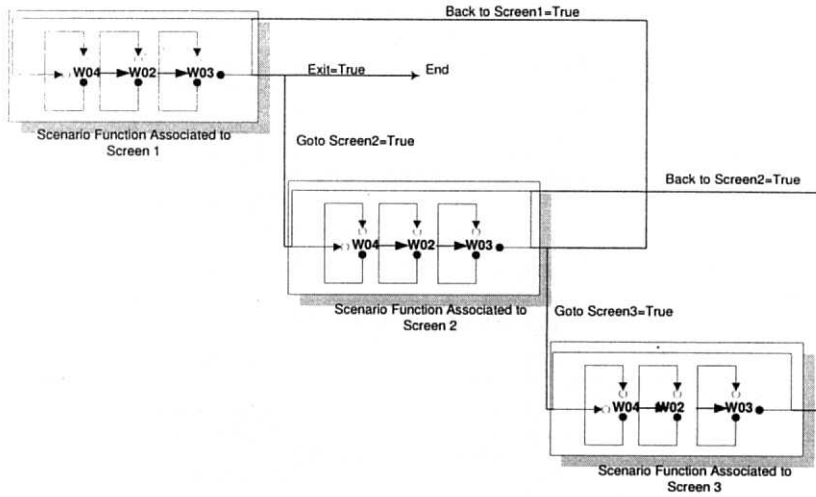


Figure 8: Processes Route Diagram.

2.4 Drawback of the Methodology

In spite of the Lyee methodology simplicity and their several positive impacts on all the steps of the software development cycle, it suffers from a major drawback which is the size of the generated code. In fact, to each word given within requirements, it attributes several memory areas. For more details about the exact amount of memory consumed by each word, reader can refer to [6, 7].

In the remain of this paper, we show how static analysis techniques can help to produce Lyee programs that run faster, consume less memory space and other better qualities.

3 Static Analysis on Lyee Requirements

Software static analysis [1, 4] means generally the examination of the code of a program without running it. Experience has shown that many quality attributes of specifications and codes can be controlled and improved by static analysis techniques. Among others, they allow to make program run faster, use less memory space and to find its bugs. Applied on requirements, static analysis allow also to find out logic errors and omissions before the code is generated and consequently they allow the user to save precious development and testing time.

The purpose of this section is to pinpoint some static analysis techniques that could improve the qualities of the Lyee requirements and their generated codes.

3.1 Optimization

Intuitively, the optimization of a program consists generally in introducing a series of modifications on it with the aim of reducing the size of its code, the time of its execution, the

consumed memory, etc. Obviously, the optimization of a given code is a strongly desirable objective, however this spot should not in any case modify the semantics of the initial program.

3.1.1 Classical Optimizations

In this section we give a brief recall of some classical optimization techniques [3, 8] and the impact of their use on the consumed memory and the execution time of Lyee programs.

- **Constant Propagation:** This simple technique consists in the detection of constants in the program, the propagation of their values along expressions using them, and finally the destruction of these constants (e.g., see Table 4).

Table 4: Constant Propagation.

Word	Definition	Condition	I/O	...
a	5			...
b	$a+3*5$...
d	$e+b*a$...
e			Input	...

Word	Definition	Condition	I/O	...
d	$e+20$...
e			Input	...

Before Constant Propagation

After Constant Propagation

- **Pattern Detection:** A pattern is a sub-expression that is repeated many times in a program. This means that each sub-expression will be computed many times. Therefore, if patterns are present in requirements, we can generally reduce the execution time of their associated code by replacing each one of this pattern by a temporary variable in which the sub-expression will be computed only one time. Table 5 gives an example where the sub-expression $b*c$ is a pattern.

Table 5: Pattern Propagation.

Word	Definition	Condition	I/O	...
a	$b*c+5$...
e	$a+b*c+1$...
d	$e+b*a$	$b*a>2$...

Word	Definition	Condition	I/O	...
t	$b*c$...
a	$t+5$...
e	$a+t+1$...
d	$e+t$	$t>2$...

Before Pattern Propagation

After Pattern Propagation

Let us now discuss how the use of these simple and classical optimization techniques can improve the memory space and the execution time of the Lyee generated codes. It is a well known fact that these optimization techniques are implemented in almost all available compilers. Furthermore, since Lyee generates generally a code in high level programming language such as Cobol, then one may conclude that once the Lyee high level code is generated, the compiler used to produce the low level code will do these optimizations. However a deep study of this problem shows that this conclusion is not totally true. In fact, the way used by Lyee to generate codes may complicate the task of the compiler when looking for

these classical optimization. To confirm that, we have written two programs in C that implement simple requirements. We have given to one of these program a structure similar to the one generated by the LyeeAll tool and the second a usual structure. After a compilation, with optimization options, of the two programs we have discovered that within the program having a Lyee structure the compiler has not been able to apply the constant propagation technique, however this optimization has been successfully done within the second program. We have concluded that within the Lyee methodology it is more beneficial and easier to use these optimization techniques before the code generation, i.e. once requirements are given by the user.

3.1.2 Optimization by Ordering Predicate Vectors

As stated before, within the Lyee methodology the order in which the user enters the statements of his requirements has no effect on the semantics (the result of the execution) of the program associated to them and this fact is one of the big contributions of this methodology. Nevertheless, the order in which we create the predicate vectors associated to these statements may have a considerable impact on the efficiency (execution time) of the generated code. Let's explain more this issue with a concrete example. Suppose that we have the requirements given in Table 6.

Table 6: Lyee Requirements: Example I.

Word	Definition	Condition	Input/Output	...
a	b + c + d		output	...
b	d*c			...
c	d+3			...
d	5			...

Suppose also that the generated predicate vectors of the pallet W'04 associated to these requirements are ordered as shown in Table 7 (a).

Table 7: W'04 Pallet Associated to the Example I.

Pallet	Program	Pallet	Program
W'04	Call S4	W'04	Call S4
	Do		Do
	Call L4_a		Call L4_d
	Call L4_b		Call L4_c
	Call L4_c		Call L4_b
	Call L4_d		Call L4_a
	while a fixed point is not reached		while a fixed point is not reached
	Call O4		Call O4
	Call R4		Call R4
(a) Not-Sorted Predicate Vectors		(b) Sorted Predicate Vectors	

Now let's briefly discuss the execution time required by this program. Once the initialization vector (S4) is executed, the program attempts, in the first iteration, to give a value to the word 'a'. This attempt will fail since the calculation of the word 'a' depends on the word

'b' which has not yet been calculated. Therefore, in this first iteration, except the word 'd', the attempt of giving a value to any word will be unsuccessful. In the second iteration, the program will succeed to attribute a value to the word 'c'. In the third iteration, the value of the word 'b' will be calculated and finally in the fourth iteration the value of the word 'a' will be found. To sum up, this program needs 4 iterations to calculate all the words.

However, if we replace the program given in Table 7 (a) by the one given in Table 7 (b), the number of the iterations needed to attribute values to all the words will drastically decrease. In fact, in only a single iteration, the program will succeed to calculate all the specified words. Hence, we conclude that the order in which the predicate vectors are generated may have a deep effect on the execution time. Consequently, it will be beneficial to tell the tool that generates code from requirements (LyeeAll) how to order the predicate vectors to reduce the execution time. Fortunately, the best arrangement of the predicate vectors can be automatically and statically generated. In fact, a simple analysis of the Def/Use of each statement given within the requirements is enough to know the best arrangement of predicate vectors. Let $s = (Id, Exp_d, Exp_c, \dots)$ be a statement where Id , is the identifier of the defined word, Exp_d the expression that define it and Exp_c its precondition. Suppose also that Exp_c , Exp_d and Id respect the following BNF grammar:

$$\begin{aligned} Exp &::= Val \mid Id \mid Exp \ Op \ Exp \\ Val &::= Int \mid Real \mid Boolean \mid \dots \\ Id &::= a \mid b \mid \dots \\ Op &::= + \mid - \mid * \mid / \mid \vee \mid \wedge \mid \dots \\ &\vdots \end{aligned}$$

where Int denotes natural numbers, $Real$ denotes real numbers and $Boolean$ denotes true and false values. Therefore, the Def/Use functions can be formally defined as follows:

$$\begin{aligned} Def(s) &= \{Id\} & Use(Id) &= \{Id\} & Use(Val) &= \emptyset \\ Use(s) &= Use(Exp_d) \cup Use(Exp_c) & Use(Exp_1 \ Op \ Exp_2) &= Use(Exp_1) \cup Use(Exp_2) \end{aligned}$$

For instance, for the requirements given in Table 6, we have the following four statements and their associated Def/Use:

Statement	Def	Use
$s_1 = (a, b + c + d, , output, \dots)$	$Def(s_1) = \{a\}$	$Use(s_1) = \{b, c, d\}$
$s_2 = (b, d * c, , \dots)$	$Def(s_2) = \{b\}$	$Use(s_2) = \{d, c\}$
$s_3 = (c, d + 3, , \dots)$	$Def(s_3) = \{c\}$	$Use(s_3) = \{d\}$
$s_4 = (d, 5, , \dots)$	$Def(s_4) = \{d\}$	$Use(s_4) = \emptyset$

In addition, we introduce a partial ordering relation, \preceq , between statements as follows: $s \preceq s'$ if $Def(s) \subseteq Use(s')$. According to this definition, the statements s_1 , s_2 , s_3 , and s_4 are ordered as follows: $s_4 \preceq s_3 \preceq s_2 \preceq s_1$

Finally, we can define the ordering over the predicate vectors as follows: If a statement that define a word a is lower than another that defines a word b , then the predicate vector $L4_a$ has to appear before the predicate vector $L4_b$ in the pallet $W04$ and the predicate vector $L3_a$ has to appear before the predicate vector $L3_b$ in the pallet $W03$. We conclude that the best way of arranging the predicate vectors of the word given in Table 6, is the one given in Table 7 (b).

3.2 Slicing

Program slicing technique goes back to Weiser [12] and it is considered as an abstraction of a program that reduce it to statements that are relevant to a particular computation. Within the traditional programming languages, slicing has long been used as a ‘divide and conquer’ approach to program comprehension and debugging (smaller programs, i.e. slices, are better understood than a large one.). It has also successfully been used to analyze many applications with respect to various goals [10] including: measuring cohesion, algorithmic debugging, reverse engineering, component re-use, automatic parallelization, program integration, and assisted verification.

Within the Lyee requirements, slicing can be helpful to analyze them with respect to different perspective. Amongst others, slicing allows us to answer the following questions:

- 1. What are the statements that contribute directly or indirectly in the definition of a given word?
- 2. What are the independent parts of requirements that may generate subprograms that we can run in parallel?

Having an automatic tool helping us to answer the first question is very useful to understand and maintain Lyee software (requirements). In fact, when the number of statements given in the requirement is important (hundred of lines), going through them to look what definition depends on what others in order to understand or/and to maintain the software, becomes a hard task and error-prone if it does not done carefully. Formally, the slice that contains statements contributing in the definition of a word w , denoted $\text{Slice}(w)$, is defined as follows:

$$\text{Slice}(w) = \{s \in S \mid s \preceq s_w\}$$

where S is the set of statements given in the requirements and s_w is, the statement that define the word w .

Table 8: Lyee Requirements: Example II.

Word	Definition	Condition	Input/Output	...
a	b + c		output	...
g			input	...
c			input	...
d	c/g	$g \neq 0$...
e			input	...
b	4*c			...

For instance, given the requirement of Table 8, $\text{Slice}(a)$ contains the statements given in Table 9 (a).

The second question concerns the independent parts of requirements. Looking for those independent parts of a given requirements is another “divide-and-conquer” technique useful to both understanding the program and to its automatic parallelization. For instance, from the requirements given in Table 8, we can automatically generate two independent slices, $\text{Slice}(a)$ and $\text{Slice}(d)$, as shown in Table 9.

Table 9: Slicing.

Word	Definition	Condition	I/O	...	Word	Definition	Condition	I/O	...
a	$b + c$		Output	...	g			Input	...
c			Input	...	d	e/g	$g \neq 0$...
b	$4 * c$...	e			Input	...
(a) Slice(a)					(b) Slice(d)				

3.3 Debugging Requirements

It is more beneficial that requirement bugs are detected and communicated to the user before the code is generated. As shown in the sequel, source of errors and their nature are various.

3.3.1 Dead Statements

A statement is considered as dead if it will never be executed. Dead statement could be due to many causes. One of the most known is the presence of contradictory preconditions within statements. In fact, if the precondition associated to a given statement is always false, then this statement is without meaning and consequently the predicate vectors associated to it will never be completely executed. This fact reflects generally a specification error and has to be communicated to the user. To detect this kind of dead code, we have only to analyze preconditions associated to statements. If it is possible to statically prove that the preconditions associated to a given statement is false (notice that it is not necessary to have the value of all the words used in a condition to evaluate it e.g. the condition $\Phi \wedge \neg \Phi$ is always false independently from the value of Φ), then this statement is dead. Furthermore, all the other statements that use a dead statements are consequently dead. More formally, if a statement s is dead, then each statement s' such that $s' \preceq s$ is also dead.

3.3.2 Cyclic statements

If requirements contain a statement s such that $\text{Def}(s) \subseteq \text{Use}(s)$, then the Lyee generated code associated to this statement can not contribute to define any word. It is another kind of a dead statements (the predicate vector associated to this statement will be superfluous). When $\text{Def}(s) \subseteq \text{Use}(s)$, we say that we have a cyclic with a length equal to zero. Cycle length may however be bigger than zero. For instance, if $\text{Def}(s) \subseteq \text{Use}(s')$ and $\text{Def}(s') \subseteq \text{Use}(s)$, then we have two dead statements and a cycle length equal to one. More generally, if $s \prec s'$ and $s' \prec s$, then the two statements s and s' are dead.

3.3.3 Incomplete Statements

A given requirements is said to be incomplete if it contains some incomplete statements, i.e., the word defined within the statement cannot be calculated from the inputs. More formally, a statement s is said to be completely defined if it respects one of the following conditions:

- $\text{Def}(s) \subseteq \{\text{input words}\}$.
- $\forall s' \mid s' \preceq s: s'$ is completely defined.

Notice also that incomplete requirements could be due to some missing information when specifying process route diagrams which may lead to one or more dead scenario functions. Recall that a process route diagram contains many interconnected scenario functions. Obviously, one of this scenario function has to be executed first and from which the control is passed to the others according to the links given by the process route diagram. Therefore, if one of the scenario function is not directly or indirectly linked to the first executed scenario function, then it corresponds to a dead code.

3.3.4 Superfluous Statements

Intuitively, a statement is said to be superfluous, if it does not contribute directly or indirectly to the generation of an output word. More formally, a statement s is said to be superfluous if it satisfies the two following conditions:

- $\text{Def}(s) \not\subseteq \{\text{output words}\}$.
- $\forall s' \mid \text{Def}(s') \subseteq \{\text{output words}\}: s \not\sqsubseteq s'$.

Obviously a superfluous statements have to be signalled to the user whatever are the raisons of their presence (specification error, introduced for later use, etc.) and the generated code has not to deal with them.

3.4 Typing

Type systems [2, 11] have largely been used to statically guarantee some dynamic well-behavior properties of programs. They allow to detect at compile-time frequent cause of errors during the execution of program. Typing techniques has also successfully been used (see [11]) to ensure that the developed software respect some security issue.

In this section, we show how typing techniques can be used to analyze Lyee requirements (to detect errors related to the association of types to words) and simplify them (automatically generate the types of the intermediate and the output words from the types of the input ones). In addition, we show how the Lyee methodology can be easily extended to deal with security issue related to software development (e.g. some sensitive information will not be leaked by the software).

The typing technique involves generally the use of the following ingredients:

- *Type algebra*: This part defines the different types that could be attributed to data, operators and statements. The domain of *types* is inductively defined as follows:

$$\tau ::= \text{int} \mid \text{real} \mid \text{bool} \mid \text{Ok} \mid \dots \mid \tau_1 \times \tau_2 \mid \tau_1 \longrightarrow \tau_2$$

The type *Ok* will be reserved for statement and the term $\tau_1 \longrightarrow \tau_2$ will be used to associate a type to each predefined operator that take parameters of type τ_1 to values of type τ_2 . For instance the operator \wedge has $\text{bool} \times \text{bool} \longrightarrow \text{bool}$ as a type.

Besides, we define an ordering relation \sqsubseteq between types to express the fact that some of them are more general than others. For instance $\text{int} \sqsubseteq \text{real}$.

Table 10: Typing System

(Val)	$\frac{\tau \sqsubseteq \text{TypeOf}(v)}{\mathcal{E} \vdash v : \tau}$	(Id)	$\frac{\tau \sqsubseteq \mathcal{E}(x)}{\mathcal{E} \vdash x : \tau}$
(Pair)	$\frac{\mathcal{E} \vdash \text{Exp}_1 : \tau_1 \quad \mathcal{E} \vdash \text{Exp}_2 : \tau_2}{\mathcal{E} \vdash (\text{Exp}_1, \text{Exp}_2) : \tau_1 \times \tau_2}$		
(App)	$\frac{\mathcal{E} \vdash \text{Op} : \tau_1 \longrightarrow \tau_2 \quad \mathcal{E} \vdash (\text{Exp}_1, \text{Exp}_2) : \tau_1}{\mathcal{E} \vdash \text{Exp}_1 \text{ Op } \text{Exp}_2 : \tau_2}$		
(Req)	$\frac{\mathcal{E} \vdash \text{Id} : \tau \quad \mathcal{E} \vdash \text{Exp} : \tau \quad \mathcal{E} \vdash \text{Cond} : \text{bool}}{\mathcal{E} \vdash (\text{Id}, \text{Exp}, \text{Cond}, \dots) : \text{Ok}}$		

- *Basic types*: To each primitive element of the language, we associate a type. The function *TypeOf* formally defines these associations as follows: $\text{TypeOf} = [\text{true} \mapsto \text{bool}, \text{false} \mapsto \text{bool}, \text{num } n \mapsto \text{int}, + \mapsto \text{real} \times \text{real} \longrightarrow \text{real}, \dots]$
- *Typing rules*: It consists of a set of rules allowing the attribution of a type to any simple or complex expression and to any statement given within the requirements. As shown in Table 10, the static semantics (typing rules) manipulates judgements of the form: $\mathcal{E} \vdash \chi : \tau$

where χ may be an expression or a statement. Intuitively, this judgement means that χ has the type τ in the environment \mathcal{E} . The environment \mathcal{E} maps types to words and it contains generally the words defined by the user within the requirements and their types. For instance, given the environment $\mathcal{E} = [c \mapsto \text{real}, b \mapsto \text{real}]$, then we prove (see Table 11) that: $\mathcal{E} \vdash b + c : \text{real}$

Table 11: Typing Proof I

$$\begin{array}{c}
 \text{(Val)} \frac{\text{real} \times \text{real} \longrightarrow \text{real} \sqsubseteq \text{TypeOf}(+)}{\mathcal{E} \vdash + : \text{real} \times \text{real} \longrightarrow \text{real}} \quad \text{(Pair)} \frac{\text{(Id)} \frac{\text{real} \sqsubseteq \mathcal{E}(b)}{\mathcal{E} \vdash b : \text{real}} \quad \text{(Id)} \frac{\text{real} \sqsubseteq \mathcal{E}(c)}{\mathcal{E} \vdash c : \text{real}}}{\mathcal{E} \vdash (b, c) : \text{real} \times \text{real}} \\
 \text{(App)} \frac{}{\mathcal{E} \vdash b + c : \text{real}}
 \end{array}$$

Notice that the main property of a type system is soundness which ensure that well-typed programs do not "go wrong" when they will be executed.

3.4.1 Errors Detection

A successful type checking of requirements guarantees that no-type errors can ever occur during the execution of their associated code. However, if we fail to give the type *Ok*, to one of the statements given within the requirements, then we deduce that there a typing error.

For instance within the specification given in Table 12, using the typing rule and the environment $\mathcal{E} = [c \mapsto \text{int}, b \mapsto \text{int}]$, we can not associate the type *Ok* to the second statement since the condition related to the definition of the word *b* is not boolean. We conclude that this second statement contain a typing error.

Table 12: Lyee Requirements: Example III.

Word	Definition	Condition	Input/Output	Types	...
c			Input	<i>int</i>	...
b	c+5	c	Output	<i>int</i>	...

3.4.2 Type Discovery

Since all the intermediate and the output words can exclusively be computed from input words, then it is possible to generate the types of the intermediate and the output words for the types of the input ones. Hence, we can simplify the task of the user and reducing his errors by asking him to specify only the types of the input words. To that end, we have first to modify the rule **Req** of the typing system as follows:

$$(\mathbf{Req}) \frac{\mathcal{E} \vdash Exp : \tau \quad \mathcal{E} \vdash Cond : bool}{\mathcal{E} \vdash (Id, Exp, Cond, \dots) : \mathcal{E}[Id \mapsto \tau]}$$

where $\mathcal{E}[Id \mapsto \tau]$ is defined as follows: $\begin{cases} \mathcal{E}[Id \mapsto \tau](Id) = \tau \\ \mathcal{E}[Id \mapsto \tau](x) = \mathcal{E}(x) \text{ if } x \neq Id \end{cases}$

Table 13: Lyee Requirements: Example IV.

Word	Definition	Condition	Input/Output	Types	...
a	b+c	b>2	Output		...
c			Input	<i>int</i>	...
b	c+5	true	Output		...

Now let us give a concrete example. Suppose that we have the requirements given in Table 13, then the type system can automatically associate the type *int* to the word *b*. In fact, since

$$(\mathbf{Val}) \frac{int \times int \longrightarrow int \sqsubseteq TypeOf(+)}{[c \mapsto int] \vdash + : int \times int \longrightarrow int}$$

and

$$(\mathbf{Pair}) \frac{(\mathbf{Id}) \frac{int \sqsubseteq [c \mapsto int](c)}{[c \mapsto int] \vdash c : int} \quad (\mathbf{Val}) \frac{int \sqsubseteq TypeOf(5)}{[c \mapsto int] \vdash 5 : int}}{[c \mapsto int] \vdash (c, 5) : int \times int}$$

we deduce by the rule **(Pair)** of the type system that: $[c \mapsto int] \vdash c + 5 : int$. Furthermore, since

$$(\mathbf{Val}) \frac{bool \sqsubseteq TypeOf(true)}{[c \mapsto int] \vdash true : bool}$$

then, from the **(Req)** of the type system that:

$$[c \mapsto int] \vdash (b, c + 5, true, \dots) : [c \mapsto int, b \mapsto int]$$

3.4.3 Security by Typing

In the sequel we show how the Lyee methodology is suitable for dealing with many other aspects of software development such as security. In fact, typing technique presented above could be easily extended to attest that a generated code satisfies some security policy such as data confidentiality and integrity when executed in a hostile environment. The idea is to allow users to explicitly attach a security label (public, secret, etc.), expressing security requirements, to each defined word, together with a security policy (e.g. the value of a secret word can not be stored in a public word). From these given information, we can use type checking techniques to automatically verify a program in order to reveal subtle design flaws that make security violations possible.

For instance, suppose that we extend the requirement by security label as shown within the statements given in Table 14.

Table 14: Lyee Requirements: Example V.

Word	Definition	Condition	Input/Output	labels	...
a	b+c	b>2	Output	public	...
c			Input	public	...
b	c+5	c>0	Output	secret	...

Suppose that our security policy forbids to affect the value of a secure word to a public one. Suppose also that the result of the addition of a secure value to another value (secret or public) has to be considered as secret. Therefore, it is clear that the example shown in Table 14 does not respect the security policy since the public word *a* has received a secret value.

To formalize this idea, we have to define the following items:

- **Labels:** they play the role of type used within the typing system and gives the names of the different levels of security that can be attached to words. These labels and the relations between them are generally defined by a Lattice [9] $\langle \mathcal{L}, \sqsubseteq, \sqcup, \sqcap, \top, \perp, \rangle$. For instance, within the Lyee requirements, we can use the following lattice:

$$\begin{aligned}
 \mathcal{L} &= \{secret, public\} \\
 \top &= secret \quad \perp = public \quad public \sqsubseteq secret \\
 public \sqcup secret &= secret \sqcup secret = secret \\
 public \sqcap secret &= public \sqcap public = public
 \end{aligned}$$

- **LabelOf:** Similarly to the function **typeOf**, the function **LabelOf** associates security label to values as follows: $LabelOf = [\text{true} \mapsto public, \text{false} \mapsto public, \text{num } n \mapsto public, \dots]$
- **Security Type System:** As illustrated in Table 15, the goal of this security type system is to reject programs that do not respect the desirable security policies.

Given the environment $\mathcal{E} = [a \mapsto public, b \mapsto secret, c \mapsto public]$, it is simple to prove that the statement $(b, c+5, c>0, \text{Output}, secret, \dots)$ respects our security policy. In fact, we have:

$$(\text{App}) \frac{(\text{Id}) \frac{\square}{\mathcal{E} \vdash c : public} \quad (\text{Id}) \frac{\square}{\mathcal{E} \vdash 5 : public} \quad public = public \sqcup public}{\mathcal{E} \vdash c + 5 : public}$$

Table 15: Security Type System

(Val)	$\frac{\Box}{\mathcal{E} \vdash v : \text{LabelOf}(v)}$	(Id)	$\frac{\Box}{\mathcal{E} \vdash x : \mathcal{E}(x)}$
(App)	$\frac{\mathcal{E} \vdash \text{Exp}_1 : l_1 \quad \mathcal{E} \vdash \text{Exp}_2 : l_2 \quad l = l_1 \sqcup l_2}{\mathcal{E} \vdash \text{Exp}_1 \text{ Op } \text{Exp}_2 : l}$		
(Req)	$\frac{\mathcal{E} \vdash \text{Id} : l_1 \quad \mathcal{E} \vdash \text{Exp} : l_2 \quad l_2 \sqsubseteq l_1}{\mathcal{E} \vdash (\text{Id}, \text{Exp}, \text{Cond}, \dots) : \text{Ok}}$		

and since:

$$(\text{Id}) \frac{\Box}{\mathcal{E} \vdash b : \text{secret}} \text{ and } \text{public} \sqsubseteq \text{secret}$$

then, we deduce, by the rule (Req), that: $\mathcal{E} \vdash (b, c + 5, c > 0, \text{Output}, \text{secret}, \dots) : \text{Ok}$.
However, the type system will reject the statement (a, b+c, b>2, Output, public, ...).

4 Lyee Requirement Analyzer

The Lyee Requirement Analyzer is a prototype that we have developed to implement a subset of the static analysis techniques previously discussed. It takes as input Lyee requirements and can give as output slices and ordered requirements suitable for the generation of optimized code by the LyeeAll tool. Besides, it can perform other requirement optimizations such as constant propagation. As shown in Fig. 9, the basic components of this prototype are the following:

- **Lexical and Syntactic Analyzers:** This part takes as input Lyee requirements and gives as output a syntactic tree commonly called intermediate representation. This new representation of requirements is the starting point of all the static analysis techniques that we are willing to do. Furthermore, when parsing the Lyee requirements, lexical or syntactic error can be detected and communicated to the user.
- **Flow-Based Analyzer:** Starting from the intermediate representation generated by the previous part, the flow-based analysis component generates all information related to the circulation of control and data flow from one requirement point to another. The results of these analysis consist of Control Flow Graph (CFG) and Data-Flow Graph (DFG).
- **Optimizer:** Amongst others, this component implements the constant propagation techniques and generates an ordered and simplified sequence of statement suitable for the LyeeAll tool to produce a program that running faster and consuming less memory.
- **Slicer:** This component takes as input flow information (such as the Def/Use associated to each word) generated by the Flow-Based Analysis component and one or many slicing criterion and gives as output slices that correspond to these given criterion.

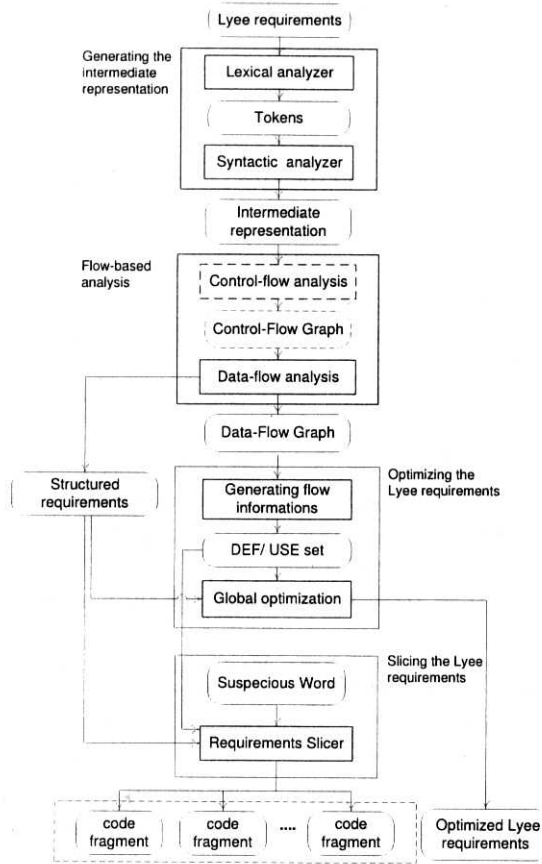


Figure 9: Lyee Requirement Analyzer Architecture.

5 Conclusion and Future Works

We have reported in this paper the use of static analysis techniques on the Lyee requirements and their impacts. First, we have presented how classical optimization techniques such as constant propagation and pattern detection can improve the execution time of the Lyee programs. We have also shown how to discover errors in requirements (dead definition, cyclic definition, incomplete or superfluous definitions). Second, we have discussed how slicing techniques can improve the understanding and the maintenance of Lyee systems. Besides, we have shown how to find out independent part of Lyee systems that can be executed in parallel using this slicing techniques. Third, we have proposed a type system allowing both the detection of typing errors and the automatic generation of types of the intermediate and output words. Forth, we have illustrated how the Lyee methodology is suitable for some extension such as security aspects. Some of the presented static analysis techniques are now implemented in a prototype called **Lyee Requirement Analyzer**.

As a future work, we want first to complete the **Lyee Requirement Analyzer** tool and

more investigate on other static and dynamic analysis techniques to improve some other aspects of the Lyee methodology.

6 Acknowledgment

We would like to address our thanks to Pr. Hamid Fujita for his support and encouragement during this research.

References

- [1] M. Bozga, J. C. Fernandez, and L. Ghirvu. Using static analysis to improve automatic test generation. pages 235–250. 2000.
- [2] L. Cardelli. Type systems. *Handbook of Computer Science and Engineering*, Chapter 103. CRC Press, 1997.
- [3] T. HENNING. *Optimization Methods*. Springer-Verlag, 1975.
- [4] S. Muchnick. *Compiler Design Implantation*. Morgan Kaufman Publishers, California, 1999.
- [5] F. Negoro. Principle of Lyee software. *2000 International Conference on Information Society in 21st Century (IS2000)*, pages 121–189, November 2000.
- [6] F. Negoro. *Introduction to Lyee*. The Institute of Computer Based Software Methodology and Technology, Tokyo, Japan, 2001.
- [7] F. Negoro and I. Hamid. A proposal for intention engineering. *5th East-European Conference Advances in Databases and Information System (ADBIS'2001)*, September 2000.
- [8] S. PANDE and D. P. AGRAWAL. *Compiler Optimizations for Scalable Parallel Systems : Languages, Compilation Techniques, and Run Time Systems*. Springer-Verlag, 2001.
- [9] D. E. Rutherford. *Introduction to Lattice Theory*. Hafner Publishing, New York, 1965.
- [10] F. Tip. A survey of program slicing techniques. *Journal of Programming Languages*, 3(3):121–189, September 1995.
- [11] D. Volpano, G. Smith, and C. Irvine. A sound type system for secure flow analysis. *Journal of Computer Security*, 4(3):167–187, 1996.
- [12] W. Weiser. Program slicing. *IEEE Trans Software Eng.*, pages 352–357, July 1984.

EFLE : An environment for generating lingware systems code from formal requirements specification

Bassem BOUAZIZ, Bilel Gargouri, Mohamed Jmaiel, Abdelmajid Ben Hamadou
{Bassem.Bouaziz, Bilel.Gargouri}@fsegs.rnu.tn
Mohamed.Jmaiel@enis.rnu.tn
Abdelmajid.BenHamadou@isimsf.rnu.tn
LARIS Laboratory FSEG-SFAX B.P. 1088 - 3018 TUNISIA

Abstract. This paper presents an Environment for Formal Lingware Engineering (EFLE) that supports formal specification and verification. The aim of EFLE is to generate lingware systems code from formal requirements specification. This environment provides interfaces enabling the specification of both linguistic knowledge and functional aspects of an application. Linguistic knowledge are specified with the usual grammatical formalisms, whereas functional aspects are specified with a suitable formal notation. Both descriptions will be integrated, after transformation, in the same framework in order to obtain a complete requirements specification. The obtained requirements specification will be refined until obtaining an executable program.

1. Introduction

In spite of the fact that applications related to Natural Language Processing (NLP) are widely used in several industrial domains, such as marketing, safety systems, security systems, etc., their development is until now based on classical tools and use ad-hoc methods. Indeed, the lingware developers make use of their know-how and their domain master's. They use several programming languages (i.e., Prolog, C++, Lisp, etc.) and varied approaches (i.e., classic, object oriented, etc.). In a study of some NLP application developments at all levels (i.e., lexical, morphological, syntactic, semantic and pragmatic) [7,21] we observed an almost total absence of methodologies that cover the whole software life cycle. The application of formal concepts is restricted, in most cases, to the linguistic knowledge representation. Generally, only algorithmic notation has been applied to describe functional aspects. Few are those who use a high level formal specification language [13,25].

In the other hand, formal methods (i.e., VDM [3], Z [22], CSP [12], CCS [18], etc.) have made their evidences in specifying and developing several safety critical systems where reliability is indispensable, such as air-traffic control systems [11], safety systems of nuclear reactors [1], operating systems and communication protocols [14]. Nowadays, formal methods provide environments and tools enabling the development of provably correct software products. These tools are, generally, based on verified refinements that transform an abstract specification into an executable program.

Starting from these advantages, we investigated the application of formal methods in the lingware development process, in order to provide solutions for the general problems indicated above. Our approach is based on a unified (or a pivot) specification language (a

¹Lingware : software related to the Natural Language Processing (NLP).

formal method notation) that supports the description of both linguistic knowledge and related treatments [8]. In this paper, we present an Environment for Formal Lingware Engineering (EFLE). This environment provides two interfaces. The first one is used to introduce linguistic knowledge. The second is used to specify functional aspects. Also, it offers tools that enable the transformation of knowledge description into the pivot specification language and their integration with functional specification, according to an appropriate technique. The obtained requirements' specifications are then refined until obtaining an executable code.

This paper is organised as follows. First, we introduce the formal approach that we proposed for lingware specification [8]. Thereafter, we present EFLE which is associated with this approach. Precisely, we indicate, for all components of this environment, the role, some related tools and an overview of the functional aspects. Finally, we make a parallel between our environment and previous works in the lingware engineering area.

2. A formal approach for lingware specification

The solution that we propose for the specification and the development of lingware is based on an approach that profits from formal methods' advantages (i.e., VDM, B, etc.). These methods have made their evidences in the specification and the development of software in general. The idea is, therefore, to use them in the natural language context while providing solutions for the problem of the dichotomy data-processing and the integration of the main existent formalisms of linguistic knowledge description.

The major problem, that we have to deal with, consists of how to represent various linguistic knowledge descriptions, initially done with different formalisms, within a unique notation. The investigation of the formal descriptions of the main existent formalisms [17,21] led us to bring out their representations within a unique formal language while ensuring the equivalence between the original and the resulted descriptions.

The expressive and unified notation (i.e., the specification language), associated to the formal method, represent a pivot language for linguistic knowledge descriptions. In addition, we can use the same language to specify, in a unified framework, both data and processing. Doing so, we can apply the development process associated to the retained formal method.

Therefore, we propose, for simplicity and convenient reasons, to acquire linguistic knowledge descriptions with the initial formalisms then realise a passage to the pivot representation by using some syntactic transformation rules as shown by Figure 1. Note that these rules preserve the semantic aspect of initial description knowledge.

This facilitates the user task, since he doesn't need to deal with the representation of the different linguistic descriptions in the pivot language. The initial formalisms for the linguistic knowledge description remain more convenient to manipulate. The use of the pivot representation of linguistic knowledge remains transparent to the user.

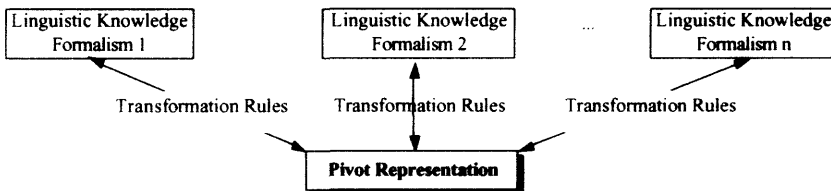


Figure 1 : Integration of the main existing formalisms for linguistic knowledge description.

Consequently, the integration of both data and processing in a unique specification, as indicated in Figure 2, can be realised once these two types of specifications were presented with the same notation. Integration allows to solve the dichotomy data-processing problem by applying formal proofs on integrated specifications.

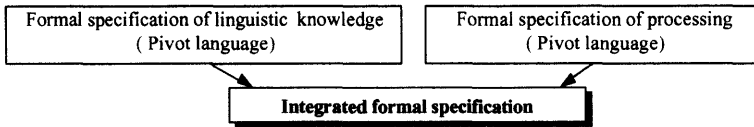


Figure 2 : Integration step of data and processing within a unified specification.

The integration of data and processing within a unified formal specification allows to apply the development process associated with the retained formal method, which is based on formal refinements and transformations of the requirements (abstract) specification.

The Figure 3 below, presents the use of the refinement and proof correctness process applied on the unified specification in the pivot language.

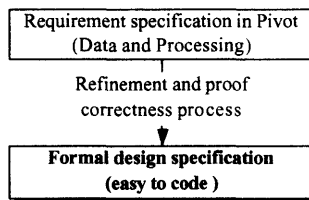


Figure 3 : Refinement and correctness proof processes of the unified formal specification in pivot language.

In this section, we presented the formal approach that we proposed to specify lingware. This approach offers solution to the majority of the lingware engineering problems. However, we can largely benefit from this approach when we develop an associated environment and implement tools that support it.

3. EFLE for generating lingware systems code

3.1. General presentation

The main purpose of EFLE is to assist lingware engineers, especially at the first steps of development. This environment proposes to generate the code of a lingware systems starting from its requirements' specifications. Accordingly, our environment provides a user-friendly interface enabling the description of the both aspects (linguistic knowledge and related treatments) separately. The functional aspect is described directly in the pivot notation, whereas knowledge aspect description is made by usual grammars' formalisms. In this way we release the user from mastering a formal language (i.e., VDM-SL), particularly for describing linguistic knowledge. Moreover, EFLE manages a dynamic library. This library includes, verified specifications of standard linguistic routines (i.e., generation, analysis, etc.) and linguistic knowledge.

Additionally, the user of EFLE can benefit from a platform of linguistic formalism evaluation that we developed separately. This platform applies a formal process based on the unified representation of the different formalisms and some evaluation criteria

[9]. This helps the user to choose the more appropriate formalism for developing a lingware.

It should be noted that we use, as pivot (or unified notation), the VDM-SL [5] which is the specification language of the formal method VDM [3] and its associated tool IFAD VDMTools.

3.2. Components of the environment

The presented environment is composed of the following four units corresponding to a provided functionality:

- Linguistic knowledge specification;
- Processing modules specification;
- Generation of the requirements' specification;
- Refinement and code generation;
- Specification library.

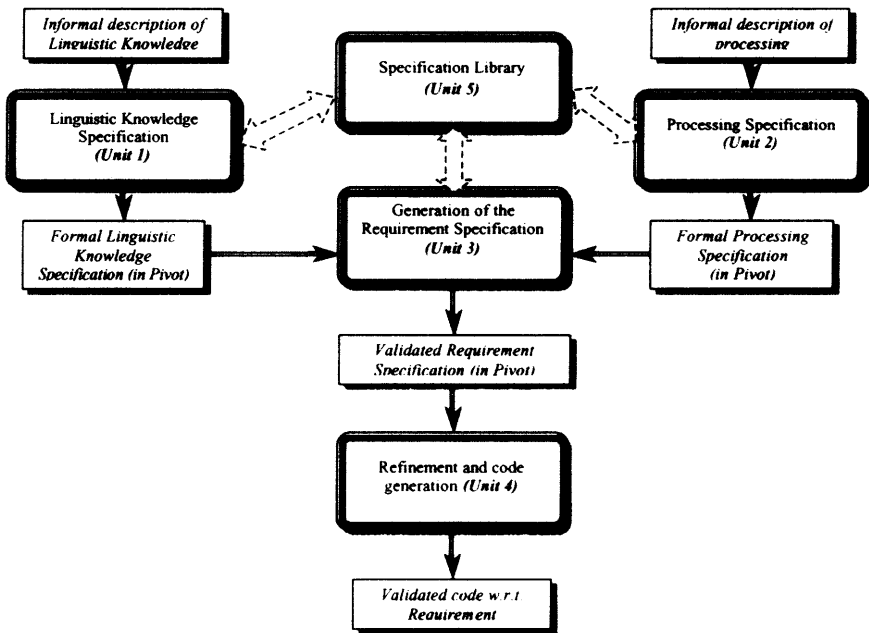


Figure 4: Interaction of EFLE units.

In figure 4, the unit3 links all the environment's units. Indeed, it uses (integrates) the two kind of specifications obtained from the unit 1 and the unit2. Moreover, the output of the unit3, constitutes the input of the unit4. These units would be described in detail in the following sections.

3.2.1. Specification of linguistic knowledge

This unit provides the necessary tools for generating a verified requirement VDM specification of the needed linguistic knowledge for the desired application. It includes the following components:

- Set of graphical interfaces;
- Set of syntactic and lexical checkers;
- Transformation tools.

The figure 5 explains how the present unit operates:

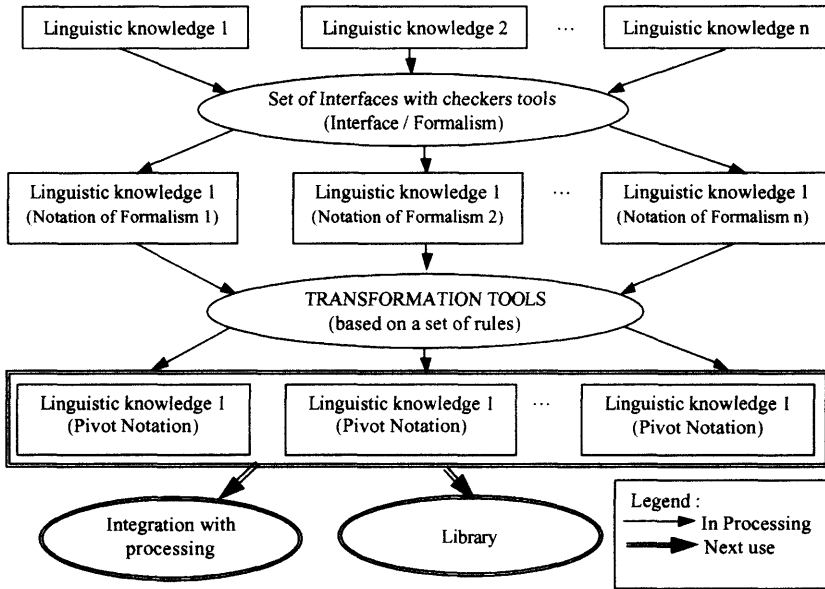


Figure 5 : Acquisition and transformation of linguistic knowledge specification.

The linguistic knowledge would be introduced using the appropriate formalisms. For each formalism we provide a suitable interface. The given descriptions will be lexically and semantically analysed until validation. Thereafter, they will be transformed in the pivot language using a specific tool based on a set of rules. This transformation guarantees the equivalence between the initial descriptions and the final ones. The resulted specification will be integrated into the processing specification of the lingware system to develop. The verified final specification may be added to the library for a possible reuse.

It should be noted that the present unit is already implemented for several formalisms (i.e., Formal Grammars, Unification Grammars, Augmented Transition Networks, etc.). Also, it is open for extensions since it is always possible to consider new formalisms by integrating tools handling them.

3.2.2. Processing modules specification

This unit concerns the specification of the functional part of a lingware system. Such a specification operates on the data types describing the linguistic knowledge. At this level, functional specifications are made in a pseudo-pivot notation. This notation is a slightly modified version of the pivot notation. Indeed, it is based on VDM-SL, but it uses usual grammatical notations, when handling linguistic data. In order to obtain the pure pivot specification we need to make links between the VDM-SL and the used grammatical notations. Such a pseudo-pivot language facilitates the user task, since he does not need to master the notation of the pivot language.

This unit is composed of the following modules as represented in the figure 6 :

- An interface allowing the formal processing specification in the pseudo-pivot notation;
- A syntactical analyser of the functional aspects specifications;
- A transformation tool that translates pseudo-notation into pivot notation.

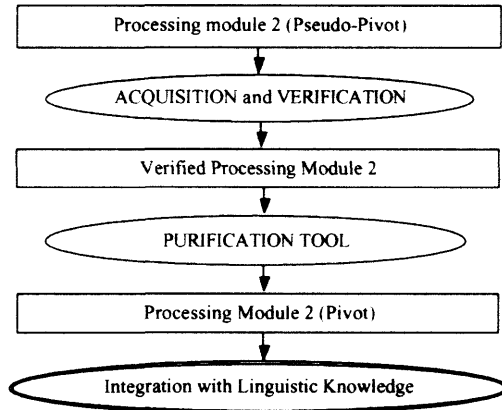


Figure 6 : Processing specification of a lingware system.

The informal descriptions of processing modules are introduced in a pseudo-pivot notation using a simple interface. This tool may be an editor associated to the retained formal method.

The specifications of the processing modules will be verified with respect to the pseudo-pivot syntactical rules, especially what concerns the identification of linguistic knowledge. This verification allows to purify these specifications to make them in pure pivot. This task consists of transforming all "not pivot" identifiers to their equivalent in pivot notation. The verification is insured since the transformation module generates only syntactically correct specification. The obtained specification will be, thereafter, used to construct the requirements specification of the lingware to develop. The semantically verification is made by tools associated to the pivot language.

3.2.3. Generating requirements' specification

The role of this unit is to construct verified complete requirements' specification of a lingware by integrating, in the unique description, the functional modules and related linguistic knowledge.

This unit is composed of the following tools (see figure 7):

- An integration tool;
- A validation tool (for requirements specification).

The following figure explains the process of constructing the proved correct requirements' specification of a lingware:

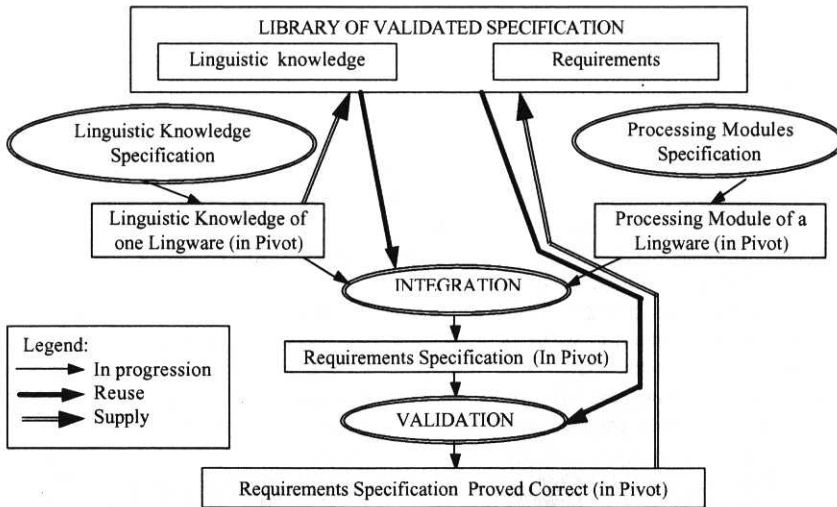


Figure 7: Construction of the requirements specification (proved correct) of a lingware system.

The resulted requirements' specifications will be verified (i.e., syntactically and semantically) using specific tools associated to the used formal method. The requirements' specification, stored in the library, can be imported (reused) and added to the current specification, in order to obtain a complete one. It should be noted that a newly constructed (verified) requirements specification can be inserted in the library for a possible reuse.

3.2.4. Refinement and code generation

The aim of this unit is to generate the code of lingware systems which satisfies the requirements specification. Currently, we make use of tools, associated to the retained formal method (i.e., SpecBox, VDMTools), which allow to ensure the refinement and the validation of specifications.

The figure below presents the unit of the specification's refinement and validation.

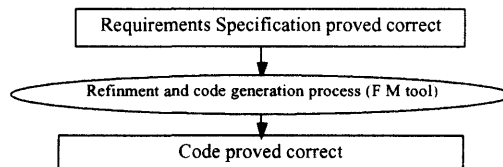


Figure 8: The refinement and code generation process.

3.2.5. Specification library

We designed an XML database for storing validated specifications (i.e. Linguistic knowledge, processing, requirements).

This unit is composed of the following tools:

- An interface for describing specifications;
- A search engine that operate on the XML database library.

The search process is based on some criteria. The user may introduce the class of specification (i.e., parsing, extracion, dictionary, etc.), the type of specification (i.e.,

Process specifications, knowledge specifications, etc.), the name of author and keywords or only one of the listed criterias. The result of process of search is given for user in XML form or in HTML.

3.3. Implementation

The present environment is already implemented in a first version using JAVA 2 Platform. It runs on a standard PC running at least the Win95 operating system. Currently, it is equipped with the most of the required tools covering the whole phases of the lingware specification process.

The figures given thereafter, illustrate the use of some tools. The first one (Figure 9) presents the main interface. It shows, a part from the menu, the shortcut of tools that are disposal at the users such that linguistic knowledge transformation, integration, purification, and validated refinement. Notably, it shows the transformation into VDM of an instance of formal grammar and its verification with the IFAD VDMtools.

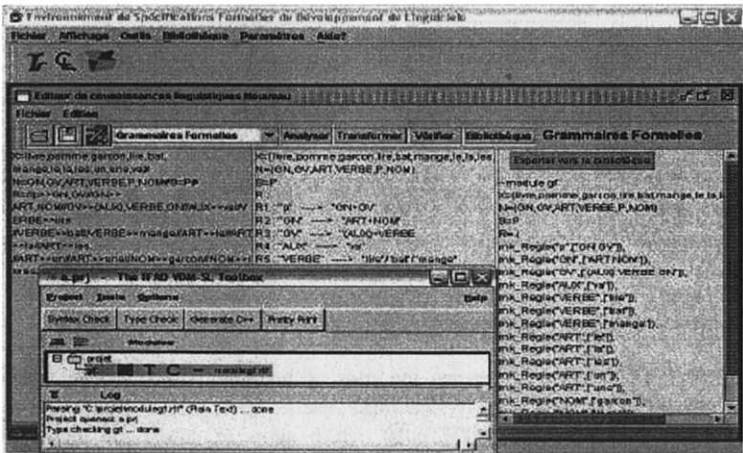


Figure 9 : Illustration of EFLE use

The figure10, illustrates the retrieval processes of specifications stored in the EFLE database library.

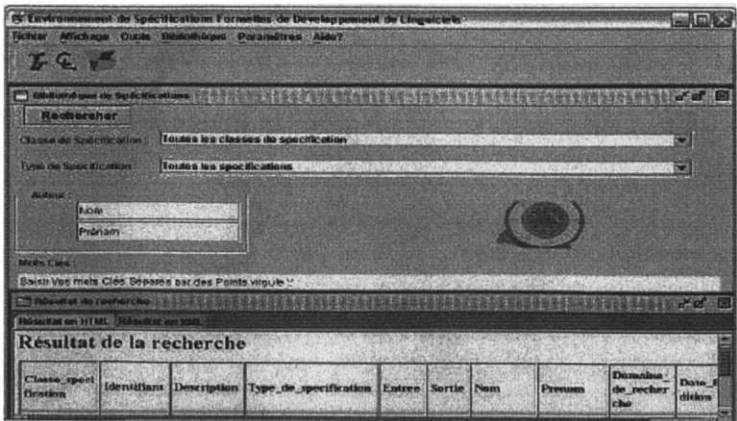


Figure 10 : Specifications retrieval in EFLE library

4. Related works

Several environments and platforms related to the lingware engineering area already exist. These environments can be classified according to their objectives. Indeed, we indicate in this section three kinds of environments. First, we distinguish some environments that are interested only in the manipulation of linguistic formalisms and consequently of grammars. Among this kind of environments, we can quote D-PATR [15] associated to the unification grammars, HDRUG[24] and HPSG-PL [20] for the HPSG grammars. Second, we make out the environments that deal with the development of specific lingware such that TIPSTER [10] and ALEP [19]. The later deals with specific applications (i.e., indexing, extraction, etc.). Finally, the third kind of environments concerns lingware systems development. We can tell that these last are relatively recent. Indeed, some of them have just coming to be proposed in a first version such that PLNLP [13], GATE[4]. Others are again projects in their first phases such that MULTTEXT/EAGLES[6,23].

These environments manipulate, in general, a particular family of formalisms for linguistic knowledge description (i.e., formalisms based on the unification) what limits the possibility to consider others types of knowledge. Furthermore, some of these environments allow the integration of data and processing using the object oriented approach, but they don't perform a formal correctness proofs. The effort is rather put on the reuse. These environments propose, generally, their proper languages for the linguistic knowledge description and processing. Some of these environments use specific concepts (i.e., task, action, object and resolution for ALEP) which requires a special training to their users.

Compared with the environments indicated above, EFLE finds its originality in the following points :

- The use of a formal method that covers all the lingware life cycle;
- The use of a pivot representation to describe linguistic knowledge. Initially, these knowledge are specified using linguistic formalisms;
- The integration of data-processing in a unified framework;
- The reuse of provably (w.r.t. initial informal specification) correct and complete specifications;
- Generation of validated code (w.r.t. requirements specification).

5. Conclusion

In this paper, we presented an Environment for Formal Lingware Engineering (EFLE) that is based on a specific approach using formal methods. This approach presents solutions to the majority of the lingware engineering problems. Also, it offers a methodology that guides the user at all steps of the development process until generating a validated code w.r.t. requirements specification.

Furthermore, EFLE offers the following advantages. First, it covers and makes automatic all the approach phases. Second, it renders easy several user tasks (i.e., some tasks are transparent to the user). Also, it is adapted to users' needs. Finally, it obeys the general principles of a software development tools, since it is user-friendly, it enables the reuse of previously verified specifications, and may be combined with other tools.

Moreover, we carried out an experimentation of specifying two real applications (i.e., a morphologic analyser [2] and a syntactical analyser[16]) using our formal approach

and profiting from the available tools. The results of these experimentation, which become strengthen our initial purpose, will be published soon.

References

- [1] Archinoff G., Verification of the shutdown system software at the darlington nuclear generating system. In Proceedings of International Conference on Control and Instrumentation in Nuclear Installations, Glasgow, Scotland, May 1990.
- [2] Ben Hamadou A., Vérification et correction automatiques par analyse affixale des textes écrits en langage naturel : le cas de l'arabe non voyellé. Thèse Es-Sciences en Informatique, Faculté des Sciences de Tunis, 1993.
- [3] Cliff B. J., Systematic software development using VDM. Prentice Hall International, 1986.
- [4] Cunningham, H., Gaizauskas R.G. & Wilks Y., A General Architecture for Text Engineering (GATE) - a new approach to language Engineering R&D. Technical report CS - 95 - 21, Department of Computer Sciences, University of Shiffeld. Available at <http://xxx.lanl.gov/ps/cmp-1g/9601009>.
- [5] Dawes J., The VDM-SL reference guide. Pitman publishing, 1991.
- [6] Erbach J.D., Manandhar S. & Uszkoreit H., A report on the draft EAGLES encoding standard for HPSG. Actes de TALN'96, 1996, Marseille, France.
- [7] Fuchs C., Linguistique et Traitements Automatiques des Langues. Hachette, 1993.
- [8] Gargouri B., Jmaiel M. & Ben Hamadou A., A formal approach to lingware development, IEA/AIE'99, May 31 - June 03, 1999, Cairo, EGYPT (LNCS, Springer-Verlag n°1611).
- [9] Gargouri B., Jmaiel M. & Ben Hamadou A., Using a formal approach to evaluate grammars, Second International Conference on Language Resources and Evaluation (LREC'2000), 31 May - 02 June 2000, Athens, Greece.
- [10] Grishman R., TIPSTER Architecture Design Document Version 2.2. Technical report, DARPA., Available at <http://www.tipster.org/>.
- [11] Heimdahl M. & Leveson N., Completeness and consistency in hierarchical state-based requirements. IEEE Transaction on Software Engineering, 6(22):363-377, June 1996.
- [12] Hoare C.A.R., Communicating Sequential Processes. Prentice Hall International, 1985.
- [13] Jensen K., Heidorn G. E. & Richardson S. D., Natural Language Processing :The PLNLP Approach. Kulwer academic publishers, 1993.
- [14] Jmaiel M., An algebraic-temporal specification of a csma/cd-protocol. In Proceedings of the IFIP WG 6.1 Fifteenth International Symposium on Protocol Specification, Testing and Verification, Warsaw, Poland, June 1995. Chapman and Hall.
- [15] Karttunen L., DPATR : A development environment for unification based grammars, COLING-ACL'86, Bonn, Germany, Auguste 1986.
- [16] Larson R.K., Warren D.S., De Lima J. F. & Sagonas K., Le système SYNTACTICA, MIT PRESS, 1995.
- [17] Miller P. & Torris T., Formalismes syntaxiques pour le traitement automatique du langage naturel. Hermès, 1986.
- [18] Milner R., A Calculus for Communication Systems, volume 90 of LNCS, Springer, Berlin, 1980.
- [19] Myelemans P., ALEP-Arriving at the next platform. ELSNews, 1994, 3(2):4-5.
ALEP home page at : <http://www.iai.uni-sb.de/alep/>
- [20] Popowich F., Available at <ftp://sfu.ca/pub/cs/nl/HPSG-PL/manual.ps.Z>.
- [21] Sabah G., L'intelligence artificielle et le langage : volume 1 et 2. Hermès, 1989.
- [22] Spivey J. M., Introducing Z: a specification language and its formal semantics. Cambridge University Press, 1988.
- [23] Thompson H., MULTTEXT Workpackage 2 Milestone B Deliverable Overview. LRE 62050 MULTTEXT Deliverable 2., 1995,
Available <http://www.lpl.univ-aix.fr/projects/multext/>.
- [24] Van Noord G., Available at <http://www.let.rug.nl/~vannoord/Hdrug/>.
- [25] Zajac R., SCSL : a linguistic specification language for MT. COLING-ACL'86, Bonn, Germany, Auguste 1986.

Author Index

Albertazzi, L.	25	Koch, G.	88
Ambriola, V.	303	Ktari, B.	375
Amon, B.	357	Levy, N.	55
Arai, O.	63	Liu, S.	75
Asari, M.	141	Malyshkin, V.	134
Ben Ayed, M.	212	Mejri, M.	375
Ben Hamadou, A.	395	Munganaze, M.	357
Bouaziz, B.	395	Nakano, T.	328
Brown, D.	239	Negoro, F.	3
Burgess, A.	254	Njabili, U.	357
Burnett, M.	227,239	Perrussel, L.	173
Charrel, P.-J.	173	Persson, A.	262
Cignoni, G.A.	303	Peters, D.	141,317
Di Sciallo, A.M.	96	Pisanelli, D.M.	125
Ekenberg, L.	357	Poli, R.	35
Erhioui, M.	375	Quantrill, M.	187
Fujita, H.	63,303	Ramdane-Cherif, A.	55
Funyu, Y.	328	Rolland, C.	155
Gangemi, A.	125	Rothermel, G.	239
Gargouri, B.	395	Salinesi, C.	196
Gruhn, V.	141,317	Sasaki, J.	328
Gustas, R.	273	Schäfer, C.	141,317
Gustienè, P.	273	Sibertin-Blanc, C.	173
Hazem, L.	55	Souveyet, C.	196
Hotaka, R.	109	Steve, G.	125
Ijioui, R.	141,317	Suzuki, H.	328
Indurkha, B.	45	Tesha, R.M.	357
Jakobsson, L.	289	v. Bochmann, G.	343
Jmaiel, M.	395	Wangler, B.	262
Johannesson, P.	357	Yamane, T.	328
Kawakami, T.	141		